UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

ALGORITMOS DISTRIBUIDOS EN CLASES DE GRAFOS Y UN NUEVO MODELO DINÁMICO DE VERIFICACIÓN DISTRIBUIDA

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MATEMÁTICAS APLICADAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

IVÁN ALONSO ZÚÑIGA TORREALBA

PROFESOR GUÍA:
IVÁN RAPAPORT ZIMERMANN
PROFESOR CO-GUÍA:
PEDRO MONTEALEGRE BARBA

MIEMBROS DE LA COMISIÓN:
JOSÉ SOTO SAN MARTIN

Este trabajo ha sido parcialmente financiado por: CMM ANID PIA AFB170001, CMM ANID BASAL ACE210010 y CMM ANID BASAL FB210005

SANTIAGO DE CHILE
2022

ALGORITMOS DISTRIBUIDOS EN CLASES DE GRAFOS Y UN NUEVO MODELO DINÁMICO DE VERIFICACIÓN DISTRIBUIDA

Esta tesis consta de dos partes. En la primera se estudia el problema del cálculo del diámetro en diferentes modelos de computación distribuida. Se inicia por describir un Proof Labeling Scheme (PLS) para resolver el problema en grafos de intervalos. Luego en el modelo `Congest` se revisa el mismo problema en grafos split y se resuelve para ciertas configuraciones especiales. Por último, en el modelo `Congest`, a partir de la estructura especial de los grafos split, se rescatan resultados del modelo `Congested Clique` como la reconstrucción de grafos de familias hereditarias.

En segundo lugar, se presenta un nuevo modelo de computación distribuida, el MID que se caracteriza por variar su estructura en el tiempo. En este modelo, se resuelven algunos problemas directos para mostrar una idea de su poder. Luego, se estudia su comportamiento para detectar subgrafos de cuatro nodos y además se encuentra una cota inferior en el problema de la detección del clique máximo. Finalmente se compara este modelo con otros modelos similares, como el modelo OLOCAL y cómo se enfrenta a problemas que se pueden resolver en esos modelos.

"Creo que es momento para otra bomba de humo
y batirme en retirada"

Babasónicos

Agradecimientos

Si bien es imposible darle las gracias a todas las personas que se merecen un espacio acá, haré mi mejor esfuerzo por lograrlo. Por quién partir, más que por mis padres Fátima e Iván, quienes han entregado su vida por mí, mis hermanos y mi hermana. Decir que esto no sería posible sin ellos se siente poco. Perdón mamá, no fui neurocirujano ni jesuita. Perdón papá, no me metí a la Santa María. Pero gracias a los dos porque sí soy feliz. Gracias Sra. Edith por todo ese amor incondicional. Gracias Seba por dejarme apuntar a la luna, gracias Diego por hacerme dormir pensando en estrellas, gracias Vale por acurrucarme cuando lloré y gracias Lucas por las cervezas, papas fritas y memes repetidos, no hubiese sobrevivido la cuarentena y este proceso sin eso.

Gracias a la gente que me acompañó en la universidad; Jipi la primera persona que me saludó en la fila de la matrícula, Tata, a quien aún no termino de devolver la mano, Camilo, Mandiola, Javi, Ojeda, Cata, Pauli, Sofi, Vale, Javi, Maca, Rubi y Tita por todos esos almuerzos y buenos momentos. Gracias a quienes fueron parte de mi paso por el DIM, Pingüino, Choco, Rola, Rondrigo, Barri y Mato, no saben lo orgulloso que estoy de ser su amigo. Gracias P. por compartir este proceso. A toda la gente que estuvo en la oficina, en especial a Emir con quien me desquité de la tesis más de una vez. Me gustaría agradecer a mi generación, pero una de las ventajas de atrasarse es que ya no tiene mucho sentido esa noción, solo destacar a todas las personas con las que estudiamos en el Open, nos tomamos un cafe o conversamos en los pasillos del departamento. Agradecer al grupo organizado de volley, que me dio vida cuando lo necesité, no solo fue un espacio de deporte para mí, si no que una comunidad donde forme grandes amistades. Gracias a todas esas personas que me regalaron alguna sonrisa en la universidad, a esas amistades que uno ni siquiera se dio cuenta cuando dejaron de ser una cara extraña en la facultad y se convirtieron en un beso y un abrazo.

Quiero agradecerle a Patricio Felmer por hacerme ver la docencia y educación de una forma distinta, por darme la oportunidad de hacer clases, conocer lo que es la investigación y participar en proyectos para quienes lo necesitan. Agradecerle a mis profesores guías, Iván Rapaport que siempre estuvo ahí con palabras de ánimo, siempre dispuesto a conversar y apoyarme en todo lo que necesité en este proceso, y a Pedro Montealegre, no podría haber pedido alguien mejor para este camino, siempre las palabras e ideas indicadas, que parecían luz cuando nos metíamos en una cueva de ideas sin salida, hasta hoy no sé cómo retribuirles todo, pero espero que estas palabras sean un comienzo. Quiero darle las gracias a Eterin, Silvia y Oscar, que me sacaron de mil y un aprietos, y a Karen quien me hizo sentir en casa después de la pandemia con el solo saludarme por mi nombre.

Quiero darte gracias Choy, porque vivir tu proceso de tesis hizo el mío más fácil aunque no estuvieras acá, gracias Chano por darme la ilusión de que se podía avanzar en la tesis en

vacaciones y gracias Pedro por enseñarme a que hay que puro seguir adelante. Gracias Gladys y don David por hacerme parte de su familia. Gracias a todas esas amistades de tantos años; Sofi, Camo, David, Carla, Sofi, Mati, Diego, Pedro, Ale, Carlos, Mascota y todas las personas que pasaron por ahí.

Ni siquiera sé por dónde empezar a agradecer a las personas que me hicieron quien soy siquiera antes de entrar a la u, a todas las personas que me acompañan desde el colegio, a todas con las que pasé incontables sábados en la mañana cuestionándome si quería seguir o no en scout. Gracias Mashka por acompañarme todas las noches que se hicieron más largas de lo que debían. Gracias Vicho por darme fuerzas. Gracias Caro e Iñaki por intentar sabotear esta tesis tantas veces. Gracias Pancho por lo que se viene. Gracias Jo por estar siempre, por hacerme soñar en grande y por permitirme escapar y apoyarme en ti cada vez que lo necesité.

Gracias a todas las personas que me dieron alguna muestra de amor, que me preguntaron cómo iba, gracias a quienes me escucharon mientras veía en sus ojos que no me entendían nada, a quienes me dijeron que nos juntáramos cuando la terminara y a quienes no mencioné, porque esta tesis es mucho más que números.

Gracias por tanto y perdón por tan poco.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

Motivación

Cuando aparecieron los computadores y estos eran gigantescos y tremendamente costosos, tenía sentido hablar sobre computación con una perspectiva centralizada, donde la máquina que teníamos a disposición era la encargada de computar respuestas a los problemas presentados.

Hoy en día, la perspectiva es muy distinta. Hay una inmensa cantidad de computadores que además están interconectados formando redes inmensas y complejas. Podemos pensar en un sinfín de ejemplos. Internet, las redes sociales e incluso las criptomonedas. En este contexto tiene también sentido tener una perspectiva distribuida, en que no hay un solo computador que hace todo, si no que se reparten las tareas en las diferentes máquinas de la red.

Un sistema distribuido puede ser modelado a través de grafos, donde cada ente o computador es representado por un nodo y cada arista del grafo representa los computadores con los cuales está conectado. Teniendo esto como base, surge una infinidad de preguntas a responder, tales como ¿cuántos mensajes debo intercambiar para poder saber qué forma tiene la red?, ¿qué tan lejos están dos nodos específicos?, ¿cuál es el nodo con el mayor número de conexiones?, etc. Responder este tipo de preguntas en un contexto distribuido conlleva varios problemas propios de la naturaleza del modelo: cada nodo puede tener información diferente, la cantidad de bits que se pueden comunicar entre un nodo y otro puede estar acotada, solo se pueden comunicar nodos que estén conectados.

Los problemas en un sistema distribuido se suelen dividir en dos categorías, los de decisión y los de cálculo. Los problemas de decisión distribuidos hacen referencia a verificar que el grafo de entrada (es decir, la red en sí misma) satisface un predicado específico. Aquí, los nodos de la red interactúan entre ellos intercambiando mensajes, siguiendo ciertas reglas, en una serie de interacciones. Después de estas interacciones los nodos deben entregar una decisión (aceptar o rechazar). Normalmente la regla de decisión especifica que una vez terminado el protocolo, si el predicado es satisfecho, todos los nodos deben aceptar, en caso contrario al menos un nodo debe rechazar. Por otro lado en un problema de cálculo dado un grafo de entrada (nuevamente la red en sí misma) se desea computar cierta función $f(G)$. En este caso los nodos también interactúan entre ellos intercambiando mensajes, siguiendo ciertas reglas, en una serie de interacciones. Al terminar este proceso normalmente la regla de cálculo especifica que todos los nodos conozcan el valor de $f(G)$. Es importante notar que no siempre es esta la regla que se usa, a veces basta con que un nodo calcule $f(G)$ y el resto acepte este resultado.

Uno de los principales modelos distribuidos es el modelo `Congest` bits. En este modelo se cuenta con una red de n nodos, donde cada nodo solo puede comunicarse con sus vecinos, es decir, puede enviar mensajes a través de las aristas del grafo. Este modelo tiene una restricción importante, los mensajes que puede enviar deben tener un tamaño acotado, cada uno de a lo más $\mathcal{O}(\log n)$. Por otro lado los nodos no tienen límite en las computaciones que pueden hacer, así el estudio de este modelo se centra en la congestión que se genera dadas las limitaciones existentes.

Dentro de los problemas básicos que puede ser necesario resolver en este tipo de modelo está el cálculo del diámetro, que responde a la pregunta ¿cuál es la distancia más larga entre cualquier par de nodos del grafo? El cálculo del diámetro en este contexto sería entonces un problema de cálculo. Se ejecuta un protocolo y al final de este los nodos de la red deben conocer cuál es el diámetro del grafo G. Esta es la pregunta inicial que se aborda en la presente tesis, tratando de encontrar respuestas para clases específicas de grafos.

Además de un modelo distribuido donde se presenta una red inicial que no varía en el tiempo, se pueden plantear modelos diferentes. Por ejemplo, si se imagina que se tiene una red central interconectada de computadores, en otras palabras un clique de n nodos, y a esta se le anclan diferentes estructuras que corresponden a redes más livianas en otros lugares, ¿hay formas de trabajar y descubrir información sobre estás estructuras de forma más rápida? ¿Se puede encontrar una forma de apoyarse en el poder de la red central? O quizás ¿se pueden hacer cálculos globales que se concentren en la red central, cosa que esta represente a todo el grafo?

Un ejemplo podría ser una empresa que tiene una sede central con todas sus computadoras conectadas. Con el paso del tiempo se abren oficinas fuera, o se ofrecen servicios en otros lugares, donde se instalan computadoras que se conectan a la red original (sede central). Puede que no sea posible hacer el mismo nivel de conexiones que en la sede Central por costos monetarios o limitaciones prácticos.

A medidas que se agregan computadoras en la periferia y se conectan entre ellas, por necesidad o costo, eventualmente se llegará a un punto en el que la red externa que se fue construyendo tendrá un tamaño similar al de la red central, en cuanto a número de computadoras, pero mucho más liviana en cuanto a número de conexiones entre ellas.

En este contexto se puede llegar a enfrentar ciertos problemas: saber si hay ciclos o no, de que tamaño son estos, encontrar un árbol generador, etc. La idea principal es explotar el poder de la red central para la resolución de esta problemática, heredado del trabajo con los grafos split hecho anteriormente y resolver el problema de la reconstrucción, es decir, el que todo nodo conozca la red es el camino más directo para que la red sea capaz de resolver cualquier problema.

Organización

En el Capítulo 1 se introducen conceptos fundamentales para el desarrollo del resto del trabajo. Se definen las clases de grafos en las que se trabajará, y se presentan diferentes modelos de computación distribuida que se usan en esta tesis.

En el Capítulo 2 se trabaja primero con grafos de intervalos, se ve un algoritmo para identificarlos y diferentes maneras de resolver el problema del cálculo del diámetro. Posterior a eso se hace un trabajo similar con los grafos split. Finalmente, se profundiza y generalizan los resultados encontrados para los grafos split.

En el Capítulo 3 se presenta un nuevo modelo dinámico de computación y se compara con otros, para una serie de problemas diferentes.

Resultados

Dentro de los principales resultados de esta tesis se exhibe un *proof labeling scheme* (PLS) para el problema del cálculo del diámetro en la clase de los grafos de intervalos con certificados de tamaño $\mathcal{O}(\log n)$. Otro resultado central es el desarrollo de un algoritmo sublineal para determinar el diámetro de un grafo split en el caso en que el clique y el independiente sean balanceados en el modelo `Congest`. A partir de la idea de esta solución, se plantea una generalización para grafos similares a los split para resolver problemas distintos al del cálculo del diámetro en el modelo `Congest`.

Finalmente se introduce un nuevo modelo dinámico y se obtienen varios resultados para diferentes problemas como el cálculo del clique máximo o la identificación de un conjunto independiente maximal (MIS) y se comparan con otros modelos dinámicos.

Capítulo 1

Preliminares

En este capítulo se entregan definiciones básicas, notaciones y herramientas que se usarán a lo largo del trabajo. Se comienza con conceptos esenciales y se fija la notación. Luego, se definen algunas estructuras importantes y, finalmente, se introducen modelos de computación distribuida que son parte central del desarrollo de esta tesis.

1.1. Definiciones básicas

Un *grafo no dirigido* o simplemente un *grafo* es un par de conjuntos $G = (V, E)$, donde V es un conjunto finito, llamado conjunto de vértices, y E es un subconjunto de $\binom{V}{2}$, llamado conjunto de aristas. Al conjunto de vértices de un grafo G se nota $V(G)$ y el conjunto de aristas de G se denota por $E(G)$. El número de vértices y el número de aristas de un grafo G lo notamos por n_G y m_G respectivamente. Se omiten los subíndices cuando se subentienda a qué grafo nos referimos.

Diremos que dos nodos u y v son *adyacentes* o *vecinos* si la arista uv pertenece al grafo. De manera similar, diremos que una arista e es incidente a un vértice v si $v \in e$. Notaremos por $N(v)$ al conjunto de vecinos de v y el cardinal de este conjunto es el grado de v, que denotaremos por $gr_G(v)$ o $gr(v)$ si no hay ambigüedad. Más generalmente, al conjunto de vecinos de los nodos de $U \subseteq V$ en $V \backslash U$ lo llamaremos $N(U)$. Decimos que $G' = (V', E')$ donde $V' \subseteq V$ y $E' \subseteq E$ es *subgrafo* de G.

Sea $G = (V, E)$ un grafo y $U \subseteq V$. Se denota $G[U]$ al subgrafo de G inducido por los vértices de U. Dicho de otra forma, para todo $u,\ v \in U$, la arista uv pertenece a $G[U]$ si y solo si uv pertenece a G.

Definición 1.1 Conjunto Independiente
Sea $G = (V, E)$ un grafo. Un conjunto de vértices $U \subseteq V$ se dirá independiente si no existen dos vértices u y v en U tales que uv esté en E.

Definición 1.2 Grafo completo
Un grafo completo se definirá como un grafo tal que $E = \binom{V}{2}$. Es decir que entre cada par de

vértices existe una arista conectándolos. Grafo completo y clique son términos equivalentes. Se denota K_n al clique de n nodos.

Definición 1.3 Camino
Un camino es un grafo $P = (V, E)$ de la forma:

$$\begin{cases} V &= \{v_0, v_1, ..., v_k\}, \text{ con todos los } v_i \text{ distintos,} \\ E &= \{v_0v_1, v_1v_2, ..., v_{k-1}v_k\} \end{cases}.$$

Normalmente nos referiremos a un camino como la secuencia de sus vértices. Es decir $P = v_0v_1...v_k$ y diremos que el camino va de v_0 a v_k. Además diremos que este camino tiene largo $k + 1$.

Definición 1.4 Ciclo
Si $P = v_0, ..., v_k$ es un camino, y $k \geq 3$ entonces el grafo C donde $V(C) = V(P)$ y $E(C) = E(P) \cup \{v_0v_k\}$ es un ciclo de tamaño $k + 1$.

Definición 1.5 Conexidad
Un grafo G se dirá que es conexo si para todo par de vértices u, v existe un camino que comienza en u y termina en v.

Definición 1.6 Árbol y bosque
Un grafo conexo y sin ciclos es un árbol. La unión vértice-disjunta de más de un árbol se llamará bosque.

Definición 1.7 Grafo de intervalos
Un grafo $G = (V, E)$ es un grafo de intervalos si existe un conjunto F de intervalos en $\mathbb{R}$ que tiene una correspondencia 1 a 1 con los vértices del grafo de modo tal que dos intervalos de F se intersectan si y solo si los dos vértices correspondientes a esos intervalos son adyacentes en el grafo G. [1]

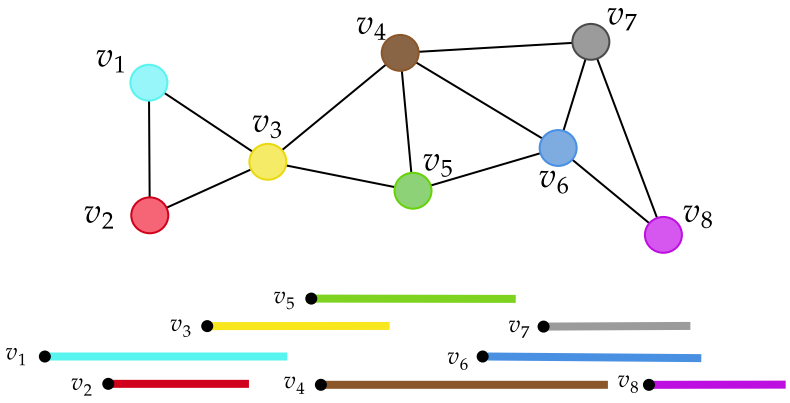

Figura 1.1: Ejemplo de grafo de intervalos y su representación de intervalos.

Definición 1.8 Grafo split
Un grafo split $G = (V, E)$ es un grafo donde V puede particionarse en dos conjuntos K e I donde $K \cap I = \emptyset$, $K \cup I = V$, K es un clique e I es un conjunto independiente. [2]

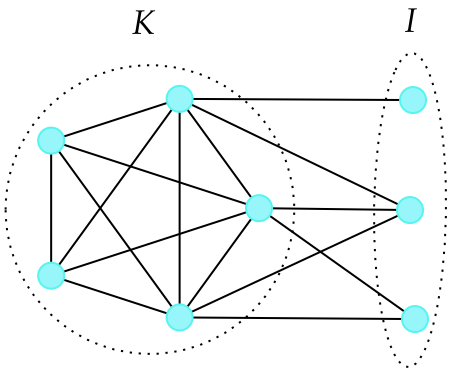

Figura 1.2: Ejemplo de grafo split, a la izquierda K que corresponde al clique e I a la derecha, el conjunto independiente.

Definición 1.9 Distancia
La distancia entre un nodo u y un nodo v en un grafo G, se denotará por $\delta_G(u, v)$ y corresponde al largo (número de aristas) del camino más corto entre u y v en G.

Definición 1.10 Excentricidad
La excentricidad de un nodo u en un grafo G se denota por $eccen(u)$ y es definida como:

$$eccen(u) = max_{v \in V}\{\delta_G(u, v)\}$$

En otras palabras, corresponde a la distancia entre u y su nodo más lejano en el grafo.

Definición 1.11 Diámetro
El diámetro de un grafo ($diam(G)$) se define como

$$diam(G) = max_{u \in V}\{eccen(u)\}$$

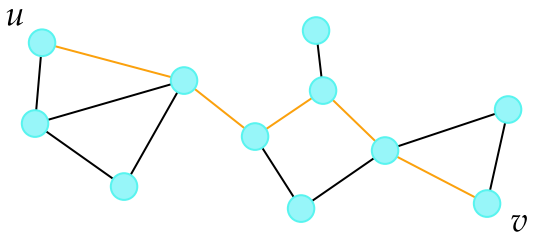

Figura 1.3: Ejemplo del diámetro de un grafo, en naranjo el camino más corto entre u y v, los nodos más lejanos.

Definición 1.12 Secuencia de grados
L secuencia de grados de un grafo G corresponde a la secuencia $(gr(v_1), ..., gr(v_n))$, donde $\{v_1, ..., v_n\} = V$ y son tales que $gr(v_1) \geq gr(v_2) \geq ... \geq gr(v_n)$.

Definición 1.13 Grafo cordal
Un grafo cordal es un grafo donde todo ciclo de largo 4 o más, tiene una cuerda. Una cuerda es una arista que no es parte del ciclo, pero que conecta dos vértices de este.

Definición 1.14 Grafo bipartito y bipartito completo
Un grafo bipartito es un grafo donde V se puede dividir en dos conjuntos independientes U y W, de manera tal que todas las aristas conectan exactamente un nodo en U y otro en W. Un grafo bipartito completo es el caso particular donde todo nodo de U está conectado a todo nodo de W.

Definición 1.15 Clases de grafos
Dos grafos G y H son isomorfos si existe una biyección $\varphi : V(G) \to V(H)$ tal que para cualquier par de vértices u y v son adyacentes en G si y solo si $\varphi(u)$ y $\varphi(v)$ son adyacentes en H. Una clase de grafos (también llamada familia de grafos) es un conjunto $\mathcal{G}$ de grafos que es cerrado bajo isomorfismos. Se denotará por $\mathcal{G}_n$ la subclase de $\mathcal{G}$ que contiene grafos de n nodos.

1.2. Modelos distribuidos

A lo largo de esta tesis se trabaja en diferentes modelos de computación distribuidas. En esa sección se definirán los modelos `Congest` y `Congested Clique` y se definirá lo que es un *proof labeling scheme* (PLS). Además se definirán tanto los protocolos deterministas, como los probabilistas.

En el contexto distribuido hay esencialmente dos tipos de problemas, los problemas de decisión y los problemas de cálculo.

Sea G un grafo conexo de n nodos. Sea $I : V(G) \to \{0,1\}^*$ una función de input que asigna etiquetas a los nodos de G acotadas polinomialmente en n. Esta función también puede ser definida para las aristas de un grafo. Sea $\mathbf{id} : V(G) \to \{1, ..., \text{poly}(n)\}$ una función inyectiva que asigna identificadores a los nodos. Una configuración de redes es una colección de tripletas $(G, \mathbf{id}, I)$. Un *lenguaje distribuido* $\mathcal{L}$ es un conjunto de pares (G, I).

Diremos que un algoritmo $\mathcal{A}$ resuelve un problema de decisión asociado al lenguaje $\mathcal{L}$ si las siguientes dos condiciones se cumplen:

- Cuando (G, I) está en $\mathcal{L}$, entonces para toda asignación de **id**'s, todos los nodos de G aceptan como resultado de $\mathcal{A}$;
- Cuando (G, I) no está en $\mathcal{L}$, entonces para toda asignación de **id**'s, existe algún nodo en G que rechaza como resultado de $\mathcal{A}$.

Para dar un par de ejemplos, definiremos los lenguajes de `LÍDER` donde se verifica si hay un único nodo en el grafo seleccionado como el líder y el lenguaje de `TWINS` que consiste en verificar si existen dos nodos cuyas vecindades sean iguales.

- $\texttt{LÍDER} = \Big\{(G, I) | I : V(G) \to \{0,1\} \text{ y } |v \in V(G) : I(v) = 1| = 1\Big\}$

- TWINS $= \Big\{(G, I)|$ existe $u, v \in G$ con $N(u) = N(v)\Big\}$

Por otro lado los problemas funcionales están asociados a una función o relación. Sea G, I e **id** definidos de igual manera que para los problemas de decisión. Se define un problema funcional como una relación $\mathcal{R}$ entre los conjuntos de pares (G, I) tales que existe una función T, cuyo conjunto de partida es V, que verifica $(G, I)\ \mathcal{R}\ T$.

Luego, decimos que un algoritmo $\mathcal{A}$ calcula $\mathcal{R}$ si para toda asignación de **id**'s todos los nodos conocen el valor de $T(v)$ tal que $(G, I)\ \mathcal{R}\ T$, una vez que el algoritmo se detiene.

Por ejemplo el problema *Minimum Spanning Tree* (MST), que consiste en encontrar el árbol generador de menor peso: tomamos las configuraciones (G, I) con $I : E \to \mathbb{N}$ como la función de pesos de las aristas. Luego el problema MST consiste en resolver la relación $(G, I)\ \mathcal{R}\ T$ donde $T(v)$ corresponde a las aristas incidentes a v en un MST de (G, I).

Las definiciones específicas varían de modelo en modelo. Por ejemplo hay modelos donde el conjunto de **id**'s está predefinido. Hay otros modelos donde también se cuenta con certificados como parte del problema. Durante este trabajo, para evitar exceso de notación y mientras haya claridad del modelo, los problemas serán enunciados sin especificar todas las funciones y relaciones. A continuación mostraremos un ejemplo con el problema MST.

MST

Input: Un grafo arbitrario G.
Output: El MST de G

Además definimos el problema principal que se trabaja en la Sección 2.

DIÁMETRO

Input: Un grafo arbitrario G.
Output: $diam(G)$

1.2.1. El modelo Congest

El modelo Congest es un modelo clásico de computación distribuida (ver [3]). El modelo Congest asume una red modelada como un grafo $G = (V, E)$ conexo. Cada nodo $u \in V$ es dotado con un identificador $\mathbf{id}(u)$ de $\mathcal{O}(\log n)$ bits de largo, tales que todos los identificadores son distintos. Los nodos son honestos, es decir, la información que entregan es confiable, y las conexiones son seguras (i.e. el modelo es libre de fallas).

Todos los nodos se inician al mismo tiempo y se procede en una secuencia sincrónica

de rondas. En cada ronda, todos los vértices envían mensajes a sus vecinos en G, reciben mensajes de sus vecinos y realizan cálculos o algoritmos locales individuales. Los mensajes enviados en una misma ronda pueden ser diferentes.

No hay límite en el poder de computación de cada nodo. Sin embargo las conexiones están sujetas a una restricción no menor: en cada ronda, no más de $\mathcal{O}(\log n)$ bits pueden atravesar cada canal. Luego, en particular, ningún nodo puede enviar más que un número constante de **id**'s a cada vecino en cada ronda. Esto hace al modelo `Congest` un buen contexto para estudiar la limitación del poder de computación de una red distribuida en el caso de una capacidad comunicación limitada, es decir, un ancho de banda pequeño.

En cada ronda del algoritmo, cada uno de los n nodos tiene que decidir si para o continúa. Un algoritmo se detiene cuando todos los nodos paran. La medida de complejidad es el número de rondas sincrónicas requeridas para que el algoritmo se detenga.

1.2.2. UCAST y BCAST

El modelo **UCAST** (Unicast) consiste en una red completa de n entidades. Cada una de estas entidades puede enviar mensajes de largo b (no necesariamente el mismo mensaje) a cada una de las otras $n-1$ entidades en R rondas. El modelo **BCAST** (Broadcast) es similar al modelo **UCAST** con la salvedad de que el mensaje enviado por cada entidad debe ser el mismo para todo el resto. En estos modelos se busca estudiar los b (largo de los mensajes) y R (número de rondas) necesarias para resolver problemas distribuidos.

El modelo **BCAST** es usual llamarlo *modelo con pizarra*. Esto viene de que el mensaje enviado por cada entidad es el mismo que debe enviar a todos, es análogo a que cada entidad escriba su mensaje en una pizarra y todos sean capaz de observar los mensajes escritos allí.

1.2.3. Modelo `Congested Clique`

El modelo `Congested Clique` introducido por Lotker, Patt-Shamir, Pavlov y Peleg [4] es un caso particular del modelo **UCAST** y a la vez similar al modelo `Congest`. Hay n nodos conectados entre sí a los que se les entrega identificadores diferentes (**id**), que se asumen por simplicidad como valores entre 1 y n. En esta tesis se considera la situación en la que el input es un conjunto G recibido de manera distribuida por los nodos de K_n. Más precisamente, cada nodo v recibe como input un vector booleano $x_v \in \{0,1\}^n$ de n bits, el cual corresponde a la función indicatriz de su vecindad en G. Es importante notar que el grafo de entrada G es un grafo arbitrario de n nodos, un subgrafo de la red de comunicación K_n, que es el grafo completo, es decir, la comunicación es desde cualquier nodo a cualquier otro nodo.

Los nodos ejecutan un algoritmo, comunicándose entre ellos en rondas sincrónicas con el objetivo de que colectivamente computen una función f que depende de G. En cada ronda del algoritmo, cada vértice envía un único mensaje de $\mathcal{O}(\log n)$ bits a cada uno de sus $n-1$ vecinos, a través de los canales de comunicación [5, 6]. En cada ronda del algoritmo, cada uno

de los n nodos tiene que decidir si para o continua. Un algoritmo se detiene cuando todos los nodos paran. Al momento de detenerse el algoritmo, todos los nodos deben conocer $f(G)$. La función f es la que define el problema a resolver y la llamamos el output del algoritmo distribuido.

1.2.4. Proof labeling schemes

Un *proof labeling scheme* (PLS) es un mecanismo distribuido para la verificación de propiedades de grafos. Más precisamente, sea $\mathcal{G}$ una familia de grafos. Un PLS para $\mathcal{G}$ es definido como un par probador-verificador (**p**,**v**), limitado a cumplir lo siguiente. Dado un grafo $G = (V, E)$ cuyos n vértices son etiquetados arbitrariamente por n identificadores (**id**) escogidos de un conjunto $\{1, ..., n^k\}$, $k \geq 1$, de rango polinomial, el probador **p** es un oráculo no confiable que le provee a cada vértice $v \in V$ con un *certificado* $c(v)$. El verificador **v** es un protocolo distribuido que ejecuta una única ronda en paralelo en todos los vértices, de la siguiente manera: Todos los vértices recolectan los certificados de todos sus vecinos y deben entregar como output "aceptar" o "rechazar", a base de los **id**, su certificado y los certificados de sus vecinos. El par (**p**,**v**) es un PLS correcto para $\mathcal{G}$ si las siguientes dos condiciones se cumplen.

Completeness. Para todo $G \in \mathcal{G}$, y para toda asignación de **id** a los vértices de G, el probador **p** (no confiable) puede asignar certificados a los vértices tal que el verificador **v** *acepta* en todos los vértices;

Soundness. Para todo $G \notin \mathcal{G}$, para toda asignación de **id** a los vértices de G, y para toda asignación de certificados a los vértices por el probador no confiable **p**, el verificador **v** *rechaza* en al menos un vértice.

La principal *medida de complejidad* para un PLS, es el tamaño del certificado asignado a los nodos por el probador.

1.2.5. Algoritmos probabilistas

Tanto en el modelo `Congest` como en el modelo `Congested Clique`, la función f define el problema a ser resuelto. Una función con conjunto de llegada 0 y 1 corresponde a un problema de decisión (como el de conectividad en el modelo `Congested Clique` [7] o la presencia de un MST en el modelo `Congest` [8]). Para otros problemas más generales f puede ser definida de manera más apropiada como una relación. Esto pasa por ejemplo si en el modelo `Congested Clique` se quiere construir un MST [9] o en la reconstrucción de un grafo [10], o en el modelo `Congest` con la coloración de distancia 2 [11]. Se recalca que en estas referencias, no se tiene la restricción de que al final del algoritmo todos los nodos conozcan $f(G)$. Por ejemplo, en el problema del MST cada nodo tiene como output un subconjunto de aristas incidentes que pertenecen al árbol.

Un algoritmo puede ser determinista o probabilista. Dentro de los algoritmos probabilistas

se distinguen dos subclases: el caso de moneda-privada, donde cada nodo lanza su propia moneda; y el caso de moneda-pública, donde la moneda es compartida entre todos los nodos. Un algoritmo $\mathcal{A}$ con error ϵ que computa una función f es un algoritmo probabilista tal que, para todo grafo G, $\mathbb{P}(\mathcal{A}$ entregue $f(G)) \geq 1 - \epsilon)$. En el caso donde $\epsilon \to 0$ a medida que $n \to \infty$, diremos que $\mathcal{A}$ computa f con alta probabilidad (w.h.p.).

Capítulo 2

Diámetro de grafos y generalización

El problema de calcular el diámetro de un grafo DIÁMETRO, que es equivalente a calcular qué tan lejos están los vértices entre ellos, es un problema de vital importancia. El conocimiento del diámetro de un grafo da una idea de qué tan conectada está la red. También entrega una cota sobre el tamaño de los caminos más cortos dentro del grafo y aún más importante, determina cuánto se demora en viajar un mensaje desde un extremo del grafo al otro.

El problema para un grafo cualquiera es bastante difícil tanto a través de un PLS como en el modelo `Congest`. Conocer el diámetro exacto de un grafo toma $\tilde{\Theta}(n)$ rondas en el modelo `Congest` [12]. Se ha demostrado que aproximaciones de $3/5 + \epsilon$ o $4/7 + \epsilon$ toman $\tilde{\Omega}(n^{1/3})$ y $\tilde{\Omega}(n^{1/4})$ respectivamente [13]. Incluso diferenciar entre si un grafo tiene diámetro 2 o 3 toma $\tilde{\Omega}(n)$ rondas [14]. Por otro lado en [15] se demuestra una cota inferior de $\tilde{\Omega}(n)$, que no es para un PLS, pero a la que se puede llegar de manera idéntica.

Este capítulo aborda cómo resolver el problema del cálculo del diámetro en ciertas clases de grafos especiales restringiendo siempre a que se recibe como promesa un grafo conexo. Primero se trabaja con grafos de intervalos y luego con grafos split. Ambas familias son subclases de los grafos cordales, una importante clase de grafos que se caracteriza por estar densamente conectada.

Luego de revisar el problema del cálculo del diámetro para los grafos split se estudiará como se pueden generalizar los métodos encontrados, para así resolver problemas más generales, como el problema de reconstrucción. Esto se hará tanto en los grafos split como en una clase similar que se define más adelante.

2.1. Diámetro de grafos de intervalos

El resultado más importante de esta subsección es la descripción de un PLS para el cálculo del diámetro en grafos de intervalos. Se inicia este trabajo con resultados menores, el primero es un algoritmo que permite regular la representación de intervalos de un grafo de intervalos. Luego se muestra un resultado fundamental para el cálculo del diámetro en estos grafos.

Finalmente se ven caracterizaciones de esta familia de grafos que permiten juntar todos los resultados y encontrar el PLS.

2.1.1. Representación de un grafo de intervalos

Se parte por un lema que será útil por 2 razones. Primero permite tener valores relevantes para el problema expresados en $\mathcal{O}(\log n)$ bits, permitiendo su fácil comunicación. Segundo evita situaciones en las que valores de los extremos de cada intervalo se pueden repetir.

Lema 2.1 *Sea G un grafo de intervalos de n vértices. Existe una representación tal que a cada vértice i se le puede asociar un intervalo $I_i = [m_i, M_i]$, tal que la colección $\{m_i\}_{i\in[n]} \cup \{M_i\}_{i\in[n]}$ de los extremos de los intervalos es igual a $[2n]$ (i.e., cada extremo es único y se encuentra en $[2n]$)*

Demostración. Sea G un grafo de intervalos con n vértices. Primero se creará una representación única de los extremos y luego estos se identificarán con $[2n]$.

Al ser G un grafo de intervalos existe una representación de este por intervalos en los reales, se referirá a esta colección de intervalos como $\{J_i\}_{i\in[n]}$, donde $J_i = [l_i, L_i]$. Supongamos que existen dos extremos L_i y L_j tales que $L_i = L_j$.

Al estar trabajando en los reales, existe un intervalo $[a, b]$ con $a, b \in \mathbb{R}$ tal que $L_i \in (a, b)$, pero ningún otro extremo (que tenga un valor diferente a L_i) está en ese segmento. Sin perdida de la generalidad se puede cambiar el valor de L_i y fijarlo como cualquier otro valor en (a, L_j), llamémoslo L'_i. Se obtiene entonces que $L'_i \neq L_j$.

Además para este nuevo valor L'_i los intervalos siguen representando el mismo grafo pues J_i sigue intersectandose con J_j y ninguna otra intersección con intervalos distintos se perdió.

Este proceso es posible repetirlo para todos los extremos iguales. De esta manera al final del proceso se tendrá que la colección $\{l_i\}_{i\in[n]} \cup \{L'_i\}_{i\in[n]}$ corresponde a $2n$ valores diferentes. Si se ordenan de menor a mayor, basta tomar un isomorfismo ϕ entre el conjunto $\{l_i\}_{i\in[n]} \cup \{L_i\}_{i\in[n]}$ y el conjunto $[2n]$ el cual respete el orden.

Así los intervalos $I_i := [m_i, M_i]$, donde $m_i = \phi(l_i)$ y $M_i = \phi(L_i)$ cumplen con lo pedido y son una representación válida de G. □

2.1.2. PLS para el diámetro de un grafo de intervalos

El diseño del PLS, que resuelva el problema DIÁMETRO en grafos de intervalos, se inicia con propiedades que caracterizan este problema, las cuales se explotan. A partir de ellas se diseña un algoritmo.

En el resto de la sección se asumirá que la representación de un grafo de intervalos es la mencionada en el Lema 2.1 a menos que se diga lo contrario, donde para un vértice v tendremos que m_v es su extremo inferior y M_v su extremo superior en el intervalo I_v asociado, además de que estos valores se encontrarán en $[2n]$.

Se definen los intervalos especiales, A y Ω de la siguiente manera (ver Figura 2.1 para referencia):

$$A = I_v, \text{ tal que } M_v = \arg\min_{u \in V}\{M_u\}$$

$$\Omega = I_w, \text{ tal que } m_w = \arg\max_{u \in V}\{m_u\}$$

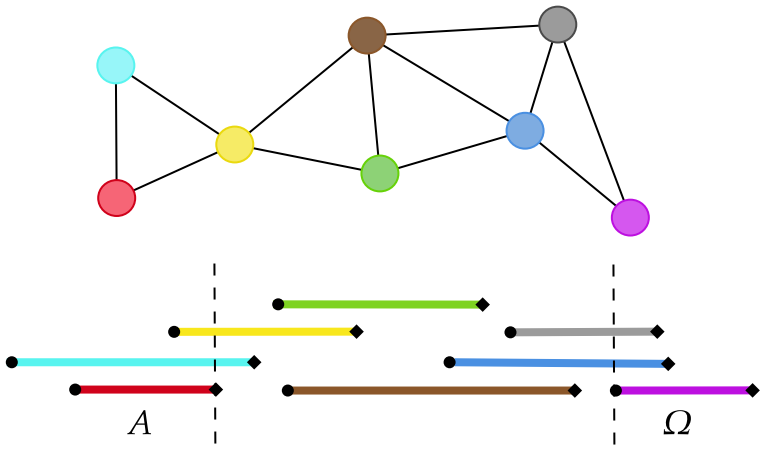

Figura 2.1: Representación de intervalos de un grafo. Los puntos negros representan los inicios de los intervalos y los rombos sus finales. Notar que no hay rombos antes del de A, ni circulos luego del de Ω.

Es decir, A es el intervalo cuyo extremo superior termina primero que los demás y Ω es el intervalo cuyo extremo inferior es el último en comenzar en la representación de intervalos reales. Esto da la idea de que son el "primer" y "último" intervalo, lo que los hace también los intervalos más lejanos, idea que formalizaremos a continuación:

Proposición 2.1 *Sea G un grafo de intervalos. Si la distancia entre los nodos v_A y v_Ω, asociados a los intervalos A y Ω, es igual a k entonces $diam(G) = k$.*

Esto es válido para cualquier representación de intervalos del grafo G, aunque A y Ω pueden variar.

Demostración. Primero veremos dos casos extremos en los que no aplica el resto de la demostración más general. Si el $diam(G) = 0$, esto quiere decir que $A = \Omega$, pues un grafo de diámetro 0 tiene un sólo nodo. Por lo tanto la distancia entre estos es 0.

En el caso en que $diam(G) = 1$ se tendrá que A y Ω son vecinos, luego la distancia entre estos será de 1.

Veamos ahora el caso más general. Se sabe por hipótesis que $\delta_G(A, \Omega) = k$, luego existen

$k-1$ vértices que forman un camino entre A y Ω que denotaremos por $\{c_i\}_{i\in\{1,...,k-1\}}$.

Estos además cumplen que:

$$v_A c_1 \in E,\ c_i c_{i+1} \in E, \forall i \in \{1, ..., k-2\} \text{ y } c_{k-1} v_\Omega \in E$$

Veamos que es posible usar este camino para conectar cualquier par de vértices. Sea $u, v \in V$, lo primero que se demostrará es que $\exists i \in \{1, ..., k-1\}$ tal que $uc_i \in E$ (el procedimiento para determinar que $\exists j \in \{1, ..., k-1\} : c_j v \in E$ es análogo).

Por contradicción asumamos que u no está conectado a ningún conector c_i, $i \in \{1, ..., k-1\}$, es decir, su intervalo no intersecta a ningún intervalo de los vértices recién mencionados.

Llamamos P al camino de c_1 a c_{k-1} obtenido al borrar A y Ω del camino de largo k que los une. Como P es un camino en un grafo de intervalos, significa que los intervalos asociados a estos vértices cubren desde m_{c_1} hasta $M_{c_{k-1}}$ de manera continua. Si no, habría una desconexión, pues para que dos vértices sean vecinos sus intervalos se deben intersectar, por lo que todos los intervalos del camino se intersectan.

Como u no intersecta a ningún intervalo asociado al camino P, significa que su intervalo no se extiende entre m_{c_1} y $M_{c_{k-1}}$, luego:

$$M_u < m_{c_1} \vee M_{c_{k-1}} < m_u,$$

en otras palabras, está antes o después del camino P.

Supongamos $M_u < m_{c_1}$ (el otro caso es análogo). Sabemos que $v_A c_1 \in E$, lo que implica que $m_{c_1} < M_A$, así:

$$M_u < m_{c_1} < M_A$$

Lo cual es una contradicción con la elección de A. Con esto se tiene que $\exists i, j \in [k-1]$ tales que $uc_i \in E$ y $vc_j \in E$. Finalmente notemos que $\delta_G(c_i, c_j) \leq k-1$ pues $\{c_i\}_{i\in\{1,...,k-1\}}$ es un camino de largo $k-2$. Así:

$$\delta_G(u, v) \leq \delta_G(u, c_j) + \delta_G(c_i, c_j) + \delta_G(c_j, v) \leq 1 + (k-2) + 1 = k$$

Con lo que concluimos que $diam(G) = k$. □

Teniendo esta propiedad y si los nodos tuviesen acceso a una representación del grafo en forma de intervalos podrían hacer directamente el cálculo del diámetro de G pues conocerían la distancia entre y Ω. Pero la verificación del diámetro sin tener conocimiento de toda la representación es más complicada. Suponiendo que cada nodo conoce solo su intervalo y un ente omnisciente entrega los nodos que conforman el camino más corto, ¿cómo podrían verificar los nodos que es efectivamente el camino más corto?

Es posible aprovecharse de la estructura de los grafos de intervalos. A partir de su representación de intervalos, para encontrar el camino minimal entre v_A y v_Ω se pueden seleccionar

los intervalos que más lejos llegan, esto es, a v_A se le asigna como siguiente conector el vecino cuyo intervalo tiene su límite superior mayor a todos los demás vecinos, a este conector de la misma manera el vecino con extremo de mayor valor y así sucesivamente hasta llegar a v_Ω. Al ser este un método glotón, la verificación se puede hacer paso a paso, es decir, no es necesario conocimiento global de la representación de intervalos.

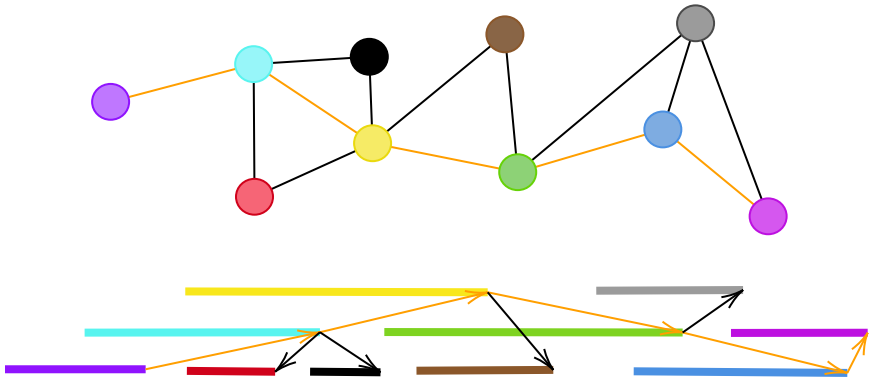

Figura 2.2: Método glotón para la elección de conectores. Las flechas que salen desde el final de cada intervalo apuntan al extremo superior de sus vecinos. Las flechas naranjas corresponden a las que llegan al extremo superior más lejano y por lo tanto son las seleccionadas para el camino.

Proposición 2.2 *En un grafo de intervalos la elección glotona de nodos genera el camino más corto entre v_A y v_Ω.*

Demostración. Supongamos que existe un camino P' más corto que el seleccionado de manera glotona P y sea u' el primer nodo de P' que no está en P. Sea v el nodo anterior a u', el cual también está en P y sea u el nodo siguiente a v en P.

Como u es el nodo que más lejos llega y que es vecino de v, eso significa que $I_{u'}$ termina antes que I_u, luego en P' es posible cambiar a u' por u y el camino sigue teniendo el mismo largo. Si se repite el proceso en cada nodo en que difieren se puede concluir que se puede cambiar todo nodo de P' por los nodos de P lo cual es un contradicción con los largos de esto caminos. □

Ahora es necesario generar una representación de los intervalos del grafo de manera en la que no sea necesaria comunicar mucha información y que además sea verificable por los nodos. El punto de partida será una importante caracterización de los grafos de intervalos, a saber, la existencia de un *camino-clique*.

Una *descomposición en árbol* de un grafo $G = (V, E)$, es un árbol T_G donde cada nodo b del árbol (a los que se llama *bolsa*) representa un conjunto de nodos $b \subseteq V$ del grafo original con las siguientes propiedades: (1) cada nodo $v \in V$ está presente en al menos una bolsa, (2) por cada arista $e = uv \in E$ existe una bolsa b que contiene a u y a v y (3) si definimos T_v como el subgrafo de T_G inducido por el conjunto de bolsas que contienen al nodo v, entonces para cada $v \in V$, el subgrafo T_v es conexo.

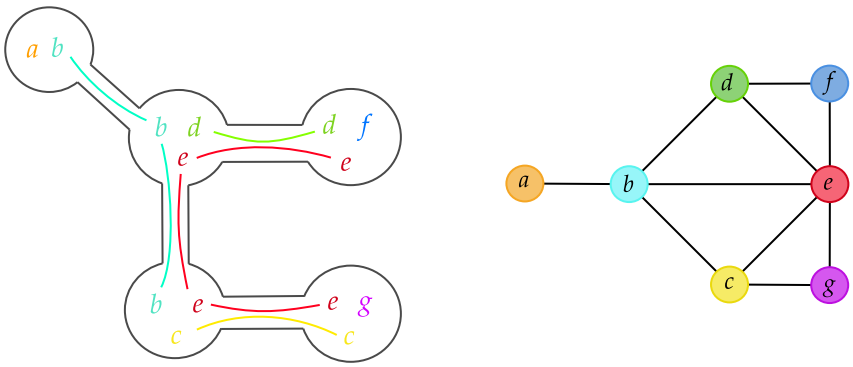

Figura 2.3: Un ejemplo de un grafo cordal a la derecha y a su izquierda su *descomposición en árbol*.

Acorde a la definición, se puede definir un $árbol-clique$ como el caso especial de una descomposición en árbol de G donde cada bolsa representa un clique maximal de G. De manera similar se define (i) una *descomposición en camino* de un grafo G como una descomposición en árbol tal que T_G es un camino, (ii) un $camino-clique$ de un grafo G es el caso especial de una descomposición en camino de G donde cada bolsa representa un clique maximal de G.

La estructura de los grafos de intervalos está determinada por sus cliques maximales, de la siguiente manera:

Proposición 2.3 ([16]) *Un grafo G es un grafo de intervalos si y solo si admite un $camino-clique$*

En [17] se desarrolla un `PLS` de una ronda para detectar tanto un grafo cordal, como un grafo de intervalos. La idea principal es seleccionar líderes de cada bolsa para que representen el $camino-clique$ y a todos los nodos entregarle el rango de cliques en los que se encuentra. Para entender la parte del trabajo que se incorporará se necesita revisar dos resultados importantes:

Lema 2.2 *[17] Para un grafo cordal G, donde T_G es un $árbol-clique$ con raíz en una bolsa ρ_T y $\{M_b\}_{b\in T_g}$ el conjunto de sus cliques maximales. Entonces es posible particionar los nodos de V en una familia $\{F_b\}_{b\in T_g}$ tal que para cualquier par de bolsas $b \neq b'$ con profundidad diferente se cumple que $F_b \subseteq M_b \backslash M_{b'}$.*

Para el caso particular de un grafo de intervalos, se tendrá un camino, por lo que las bolsas siempre tendrán distintas profundidades. De aquí se saca que en cada clique contiguo hay nodos diferentes. Esto entrega la posibilidad de escoger líderes diferentes para cada bolsa, pero estos no necesariamente estarán conectados entre si. El siguiente resultado permite corregir este problema y otras dificultades técnicas asociadas al caso de los grafos cordales en los que se tienen árboles, donde se podría dar que un líder estuviera a cargo de muchas bolsas en un mismo nivel, mas estos detalles quedan fuera de la cobertura de esta tesis.

Lema 2.3 *[17] Dado un grafo cordal G, existe un árbol − clique T tal que es posible escoger una colección de líderes $\{v_b\}_{b\in T_g}$ para cada bolsa y nodos auxiliares $\{w_l\}_{l\in T_g}$ para cada hoja en T tales que, si* ***depth****(b) es la profundidad de una bolsa b y $t(b)$ es el padre de la bolsa b en el árbol, entonces:*

- *Por cada $b \in T$, $v_b \in b$*
- *Por cada $b \in T$, $v_b v_{t(b)} \in E(G)$*
- *Si* ***depth****$(b) \neq$* ***depth****(b'), entonces $v_b \neq v_{b'}$*
- *Si $b \in T$ es una hoja, entonces $w_b v_b \in E(G)$*
- *$\{v_b\}_{b\in T_g} \cup \{w_l\}_{l\in T_g} = \emptyset$*

Teniendo lista la parte previa correspondiente a la construcción del *árbol − clique*, el siguiente resultado se centrará en generar una representación de los intervalos del grafo original a partir del *camino − clique*, por lo que juntando ambas piezas se estará a un paso del `PLS` final.

Proposición 2.4 *Sea G un grafo y P_G el camino − clique de G. Sea b_0 uno de los extremos del camino, al que nos referiremos como su inicio. Para $b \in V(P_G)$ se denotará por $d_P(b)$ la distancia al inicio del camino P_G.*

Si a cada bolsa b del camino − clique P_G le asignamos el intervalo $[d_P(b), d_P(b)+1)$ y a cada nodo $v \in V$ le asignamos el intervalo correspondiente a la unión de los intervalos de las bolsas en las que está en P_G, entonces esa representación de intervalos es en efecto válida para el grafo G.

Demostración. Para esto verificaremos que a cada nodo se le asigne un intervalo que está conectado con los intervalos de sus vecinos y no con el resto de los vértices del grafo.

Primero notemos que efectivamente a cada nodo v se le asigna un intervalo. Como P_G es un *camino − clique*, v está en al menos una bolsa. Además se cumple que P_v es conexo, luego el intervalo que se le asigna es la unión de intervalos contiguos, por lo tanto es un intervalo conexo válido.

Segundo, veamos que efectivamente a cada nodo v se le asigna un intervalo que este conectado con todos los intervalos de sus vecinos y no con los nodos que no son adyacentes a él.

Para esto notemos que como P_G es *camino − clique*, entonces $\forall uv = e \in E,\ \exists b \in P_g$ tal que $uv \in E$. Por lo tanto v comparte bolsa con todos su vecinos, luego comparte un tramo de sus intervalos, es decir, están conectados.

Veamos ahora que no se intersectan con ningún otro nodo. Supongamos por contradicción que v y w no están conectados en el grafo G pero sus intervalos si se intersectan. Esto significa que comparten una bolsa, pero cada bolsa de un *camino − clique* representa un clique maximal, por lo tanto v y w son parte de un mismo clique maximal, es decir que están conectados, lo que es una contradicción. □

Juntando todo lo anterior se puede demostrar el principal resultado de esta sección:

Teorema 2.1 *Existe un* PLS *para determinar si el diámetro de un grafo de intervalos es k con certificados de $\mathcal{O}(\log n)$ bits en una red de n nodos.*

Demostración. La primera parte del PLS consistirá en imitar el PLS de [17]. En este se escogen líderes para representar el $camino - clique$ y se verifica esta representación. Luego a partir de esto se entregarán los intervalos a cada nodo, de lo cual se inferirá el diámetro del grafo.

Asumiendo que el grafo es de intervalos, seleccionamos una colección de nodos que representarán a las bolsas b de P_G. Esto lo hacemos escogiendo un único elemento ρ_b de cada conjunto F_b de acuerdo al Lema 2.2. Además para verificar la estructura de camino se entrega a cada líder la tripleta $\langle \mathbf{id}(\rho_P), d_P(v), t_P(v)\rangle$ que indica el líder único del camino, la distancia a él y el padre en el camino de ρ_b.

Por otro lado, a cada nodo $v \in F_b$ se le entrega:

- El largo del camino P_G, junto al **id** del líder ρ_P
- Una etiqueta $F(v)$ que corresponde al identificador $\mathbf{id}(\rho_b)$ del líder en el conjunto F_b al que el nodo v pertenece, junto al tamaño de F_b.
- La distancia de la bolsa b a la raíz ρ_P en el $camino - clique$ con $v \in F_b$, dada por $\mathbf{depth}(v)$.

Para verificar la estructura, también se escoge una colección de líderes $e_{bb'}$ por cada arista $bb' \in E(P_G)$ junto a los nodos auxiliares correspondientes dados por el Lema 2.3.

Por último, para revisar el diámetro se le entregará a los nodos $\{u_i\}_{i\in[k]}$ que formen el camino más corto entre A y Ω la tupla $\langle \mathbf{id}(A), d(v, A), c(v), k\rangle$, es decir el **id** del primer nodo, su distancia a este, su padre en el camino y el largo del camino, esto dado el Lema 2.1.

Con está información los nodos pueden conocer el intervalo que les corresponde en la representación creada a partir del $camino{-}clique$, con esto intercambian mensajes y verifican:

- Las colecciones $\{\rho_b\}_{b\in P_G}$ y $\{e_{bb'}\}_{bb'\in E(P_G)}$ verifican la correctitud de la estructura (deben verificar la unicidad de hijos).
- Hay una única raíz ρ_P
- Si $v \in F_b$, entonces v verifica que los nodos con $F(v) = \mathbf{id}(\rho_b)$ formen un clique
- Si u y v son adyacentes con $\mathbf{depth}(v) \leq \mathbf{depth}(u)$, entonces v es adyacente a todos los líderes (y sus conjuntos F_b) en el camino entre $F(v)$ y $F(u)$.
- La colección $\{u_i\}_{i\in[k]}$ verificará que la estructura esté correcta y cada nodo además verificará que haya sido escogido de manera glotona.
- Los nodos A y Ω verifican que cumplan con ser el primer y último intervalo respectivamente (esto lo pueden hacer de manera local).

Si todas las condiciones anteriores se cumplen todos aceptan. Veamos ahora la correctitud del protocolo.

Completeness: Supongamos que este es un grafo de intervalos de diámetro k. Un probador honesto generará un $camino - clique$ correcto (como es demostrado en [17]) de lo cual se obtendrá una representación de intervalos válida para el grafo G como se mostró en la Proposición 2.4. Además el probador seleccionará de manera correcta los $k-1$ conectores, A y Ω, haciendo así un camino de largo k que por la Proposición 2.1 implica que el diámetro del grafo es k.

Soundness: Supongamos que el diámetro del grafo no es k. Una posibilidad es que existen dos nodos u y v tales que su distancia es mayor a k. Esto implica que el nodo asociado a A y el nodo asociado a Ω también están a distancia mayor a k. Si este fuese el caso entonces habría un camino de largo mayor a k o los nodos conectores no forman un camino (están desconectados). Ambas posibilidades son verificadas por el algoritmo mostrando que los conectores efectivamente forman un camino de largo k.

La segunda opción es que el diámetro sea menor, es decir, el protocolo entregue que el diámetro es k cuando el real diámetro del grafo es menor a k. La construcción glotona que se demuestra en la Proposición 2.2 verifica que el camino formado por los conectores entre los nodos asociados a A y Ω respectivamente sea el de menor largo posible. La verificación de una construcción glotona de los nodos resguarda que el algoritmo cometa este error.

Un último punto de fallo es que la construcción de los intervalos sea defectuosa, como la Proposición 2.4 contradice esto, el error tendría que venir desde la construcción del $clique - camino$, pero esto no es posible pues la construcción es a partir de [17]. □

2.2. Diámetro de grafos split

En esta subsección se resolverá el problema DIÁMETRO para otra clase importante de grafos, la de los grafos split. Esta clase consiste en grafos que se pueden dividir en un clique y en un conjunto independiente, es decir, una colección de nodos que se conecta de manera arbitraria al clique pero que no se conectan entre ellos. Esta clase de grafos, al igual que los grafos de intervalos, es una subclase de los grafos cordales.

Los grafos split tienen la particularidad de tener un diámetro sumamente acotado, cada nodo del clique está a distancia 1 del resto del clique y a lo más a distancia 2 de cualquier independiente, puesto que es vecino del nodo al que está anclado. Por otro lado los nodos independientes están a distancia 2 de cualquiera del clique y a lo más a distancia 3 de los demás independientes, puesto que el clique los conecta. Es decir un grafo split puede tener solamente diámetro 2 o 3. Aún así, diferenciar entre estas dos situaciones tiene un alto costo computacional.

2.2.1. Reconocimiento de grafo split en el modelo Congest

En esta subsección, mostraremos una caracterización de los grafos split que nos permite explotar su estructura y desarrollar protocolos para resolver problemas, como el del cálculo de su diámetro.

Lema 2.4 *[18] Sea $d_1 \geq d_2 \geq ... \geq d_n$ la secuencia de grados de G y sea m el valor más grande de i tal que $d_i \geq i - 1$. Entonces G es un grafo split si y solo si*

$$\sum_{i=1}^{m} d_i = m(m-1) + \sum_{i=m+1}^{n} d_i.$$

Si es este el caso, entonces los m vértices con mayor grado forman un clique máximo en G y el resto de los vértices corresponden al conjunto independiente.

Una manera de interpretar este lema para el caso en que sabemos que es un grafo split, es que los vértices del clique tendrán grado mayor o igual al tamaño del clique.

Hay casos problemáticos donde se puede dar un clique de tamaño k al que se le conectan 2 vértices u y v de manera completa. Se forma un grafo split, donde el clique máximo es de tamaño $k+1$, pero hay 2 de estos cliques. En este caso u y v se denominan *twins*, puesto que tienen exactamente los mismo vecinos. Si no fuese porque tanto a u como a v se le asignan **id**'s diferentes, no habría manera de diferenciarlos.

Los nodos twins pueden generar problemas, ¿cómo se detecta que estamos en una situación así? y aunque se supiese que hay twins y quiénes son, ¿cómo decidir que nodo es parte del clique máximo? A menos que se comuniquen a través de todo el grafo, no hay manera de coordinar esta decisión.

Por las complicaciones mencionadas muchas veces se opta por ignorar este tipo de situaciones, es decir, ignorar a los nodos twins. En [19] se desarrolla una manera de reconocer en el modelo Congest con alta probabilidad a los nodos twins, de esta manera se pueden tomar medidas para evitar estas complicaciones. El siguiente paso en el trabajo es entonces deshacerse de los nodos twins, para esto se hacen modificaciones del algoritmo descrito en [19].

Posterior a eliminar los twins, se identifica a cada nodo con su conjunto correspondiente, ya sea el clique o el conjunto independiente. Es importante notar que el caso problemático es cuando los twins son parte del independiente, puesto que ellos generan la ambigüedad de cuál es el verdadero clique máximo. Además sin siquiera saber qué nodos están en el clique y cuáles están en el independiente se puede notar una diferencia que ayudará en la demostración del próximo resultado: los vértices independientes no estarán conectados entre sí, por lo tanto los twins que son interesantes son aquellos que no están conectados.

Proposición 2.5 *Sea G un grafo split y sea $c > 0$ constante. Existe un algoritmo de 2 rondas con moneda pública en el modelo* `Congest` *que identifica a los vértices twins del conjunto independiente con un error de $1/n^c$.*

Demostración. Partiremos por simular el proceso realizado en [19] adaptándolo al modelo `Congest` y al caso de un grafo split.

Sea $n^{c+3} < p \leq 2n^{c+3}$ un número primo. Se escoge un valor $t \in \mathbb{Z}_p$ uniformemente al azar usando $\mathcal{O}(\log n)$ bits aleatorios públicos. Dado un vector de n bits $a = (a_1, ..., a_n)$, sea el polinomio $P_a = a_1 + a_2X + a_3X^2 + ... + a_nX^{n-1}$ y sea $FP(a,t) = P_a(t)$. $FP(a,t)$ se llamará el $fingerprint$ de un vector a. Claramente si dos vectores son iguales tienen igual fingerprint, y más importante, para dos vectores distintos a y b, la probabilidad de que $FP(a,t) = FP(b,t)$ es a lo más $1/n^{c+2}$ (pues el polinomio $P_a - P_b$ tiene a lo más n raíces y t fue escogido uniformemente al azar, luego la probabilidad de que t sea una raíz para $P_a - P_b$ es a lo más $1/n^{c+2}$, ver ejemplo en [20]).

Sea x_i la función característica del vecindario de un nodo i, $N(i)$ (el vector del input de un nodo). El algoritmo consiste en:

Protocolo identificación twins

1. Cada nodo envía el mensaje $m_i = FP(x_i, t)$
2. Si un nodo recibe 2 mensajes iguales le notifica al nodo con el menor **id** de ambos que es un candidato a twin y además le envía el **id** de su posible twin
3. Si un nodo es notificado de que es un candidato a twin, compara el **id** de su posible twin con los de sus vecinos, si este no calza con ningún vecino, se declara twin, si no omite.

Veamos ahora la correctitud del algoritmo. Claramente si en el input hay 2 nodos independientes con los mismos vecinos, tendrán el mismo fingerprint y no estarán conectados. El nodo de menor **id** será notificado y se identifica como twin pues no está conectado al otro nodo.

Por otro lado, dos nodos del clique no se reconocerán como twins aunque lo sean pues son vecinos. Además un nodo del independiente y un nodo del clique no pueden ser twins, ya que si lo fueran, entonces ambos serían del clique.

Luego la probabilidad de que el algoritmo haga a un nodo identificarse como twin cuando no lo es, es la probabilidad de que $FP(m_i, t) = FP(m_j, t)$ para dos nodos no conectados. Para cada par cualquiera la probabilidad de esto que esto suceda es $1/n^{c+2}$, por lo tanto juntando las probabilidades esta será menor a $1/n^c$. □

Dadas las características de los grafos split podemos obtener un resultado más general.

Corolario 2.1 *Sea G un grafo split y sea $c > 0$ constante Existe un algoritmo de 2 rondas con moneda privada en el modelo* `Congest` *que identifica a los vértices twins del conjunto independiente con un error de $1/n^c$.*

Demostración. Al estar trabajando en grafos split se cumple que el diámetro de estos es a lo más 3. Luego cualquier mensaje puede ser propagado para todo el grafo en a lo más 3 rondas. Luego se puede aplicar el siguiente protocolo:

Protocolo moneda privada
1. Cada nodo comparte su vecino de menor **id**.
2. Cada nodo comparte el **id** más pequeño que le llegó.
3. El nodo de menor **id** comparte sus $\log n$ bits aleatorios.
4. El mensaje es pasado al resto del grafo.

Veamos porque este algoritmo funciona. Lo primero es la selección del menor **id**. Al ser un grafo split, los nodos del clique en la primera ronda se enteran de quién es el vértice de menor **id** pues todo nodo u independiente está conectado a alguien en el clique, quien compartirá el **id** de u si es el menor. Una vez esto listo, para que los nodos del independiente también sean conscientes de esto, se vuelve a compartir la identificación más pequeña.

Luego de estas 2 rondas de interacción el nodo de menor **id** comparte sus bits aleatorios (su moneda privada) y este mensaje es extendido al resto de los nodos. Al tener un diámetro menor o igual a 3 este proceso toma a lo más 3 rondas. Luego todo el proceso toma un número constante de rondas, manteniendo la complejidad del resto del protocolo (identificación de twins) que es presentado en la Proposición 2.5. □

Teniendo resuelto el problema de los nodos twins es posible seguir con el trabajo de hacer que cada nodo identifique si pertenece al clique o al conjunto independiente del grafo split.

Proposición 2.6 *Sea G un grafo split, donde K es el conjunto de vértices del grafo completo máximo de G e I es el conjunto de vértices independientes que cumple que $K \cup I = V$ y sea $c > 0$ una constante.*

Existe un algoritmo probabilista de $\mathcal{O}(1)$ rondas en el modelo `Congest` *que identifica a cada nodo $v \in V$ con K o con I con un error de $1/n^c$.*

Demostración. La idea del protocolo es que los nodos comuniquen sus grados. Como se parte de la base de que es un grafo split, basta ordenar estos valores para que de manera local los nodos sepan cual es su clique maximal. Al ser un clique maximal un grafo completo, loas examinaciones locales bastan para identificarlo.

Lo primero es deshacerse de los casos complicados, es decir, los twins. Para esto se corre el algoritmo de la Proposición 2.5. Una vez listo este proceso podemos pasar al protocolo.

Protocolo identificación clique e independiente
1. Cada nodo v comparte su grado con sus vecinos.
2. Una vez cada nodo v cuenta con la secuencia de grados $d_1 \geq d_2 \geq ... \geq d_{gr(v)}$, busca el mayor índice i tal que $d_i \geq i - 1$ y lo denominará m_v.

3. Cada nodo v comparte m_v con sus vecinos.
4. Si a un nodo v le llega $m_v - 1$ veces el valor m_v, entonces se encuentra en K, si no se encuentra en I

La idea detrás, es que el valor m_v corresponde al tamaño del clique más grande en el que podría estar v, por lo que si tiene $m_v - 1$ vecinos que coinciden con esto entonces, se encuentra en el clique maximal y este tiene tamaño k. Si a un nodo le llega un valor k mayor a su m entonces pertenece al conjunto independiente.

Veamos la correctitud de este algoritmo: supongamos que se trabaja con un grafo G split con clique máximo de tamaño k, esto significa que los vértices que sean parte del clique máximo K, tendrán $k-1$ vecinos de grado al menos $k-1$, de manera que al aplicar el protocolo se reconocerán. Por otro lado al no haber twins, el resto de los nodos tendrán grado estrictamente menor. Además son vecinos de alguien en el clique, por lo que les llegará al menos un mensaje con un valor mayor o igual al de su grado, identificándose correctamente con el conjunto independiente I.

El único caso en el que podría haber un error es el que a dos nodos independientes entre si, les llegue información que indique que están conectados a un clique de tamaño $k-1$, este es el caso en el que estos dos nodos son twins, pero esto sucede con probabilidad $1/n^c$ por la proposición anterior.

A las dos rondas del algoritmo descrito es necesario sumarle las dos rondas del protocolo anterior, el que además tiene asociado un error de $1/n^c$. Por lo tanto el algoritmo es ejecutado en $\mathcal{O}(1)$ rondas con un error de $1/n^c$. □

2.2.2. Diámetro de grafos split con clique pequeño

Cuando se habla de grafos split, si bien tienen una estructura sumamente definida, hay un factor importante que los puede diferenciar entre sí: la proporción de nodos que se encuentran en el clique y los que hay en el independiente. Veremos que algunos casos son más fáciles computacionalmente que otros. Se inicia la sección con algunos casos desbalanceados más simples y luego se ataca el principal problema, el caso balanceado.

El primer y más simple caso es en el que el número de nodos del clique es constante k, lo que implica que dejando de lado los twins, hay un número limitado de combinaciones posibles.

Lema 2.5 *Sea $k > 0$. Existe un protocolo en el modelo* `Congest` *que determina* DIÁMETRO *en $\mathcal{O}(2^k)$ rondas en un grafo split G cuyo clique máximo es de tamaño k.*

Demostración. Cada vértice que no sea parte del clique tiene que estar conectado a nodos de este y a ningún otro nodo, luego hay 2^k posibilidades de conexiones (antes de que se repitan y vuelvan a aparecer twins), a la que hay que descontar la opción en la que se conecta a

todos los vértices, pues en ese caso forma un clique de tamaño $k+1$.

Luego, al haber solo $2^k - 1$ posibles nodos, toda la información puede ser transmitida en un número constante de rondas, por lo que el diámetro es determinable en un número constante de rondas. □

Para el siguiente caso se asumirá que el número de nodos del conjunto independiente I de un grafo split G es n. La segunda situación que se analizará es cuando el número de vértices del clique es $\mathcal{O}(\log\log n)$. Si bien, a diferencia del caso pasado, el número de nodos en el clique no es fijo, sigue siendo lo suficientemente pequeño como para poder ser controlado.

Lema 2.6 *Sea G un grafo split, donde K es el conjunto de vértices del grafo completo máximo de G e I es el conjunto de n vértices independientes que cumple que $K \cap I = V$. Existe un protocolo en el modelo* `Congest` *que resuelve* `DIÁMETRO` *en $\mathcal{O}(1)$ rondas si $|K| = \log\log n$.*

Demostración. De manera similar a la demostración anterior, dejando de lado nodos twins, el número de posibles conexiones del grafo es $2^{\log\log n}$, es decir, $\log n$. Luego, sigue siendo una cantidad de información que se puede comunicar en un número constante de rondas. □

2.2.3. Diámetro de grafo con conjunto independiente pequeño

Un aspecto fundamental de los grafos split es la total conexión de sus cliques, característica que hasta ahora no ha sido explotada. En los siguientes casos, donde el tamaño del clique empieza a ser más significativo, esta será la principal herramienta que se usará para determinar de manera eficiente el diámetro del grafo.

Se partirá por el caso en el que los nodos del clique dominan, es decir cuando el número de nodos independientes es a lo más $\sqrt{|K|}$. La idea será que los vértices del clique trabajen en conjunto puesto que su alta conexión hace esto más eficiente.

Proposición 2.7 *Sea G un grafo split, donde K es el clique máximo de G e I es el conjunto de vértices independientes que cumple que $K \cup I = V$. Existe un protocolo en el modelo* `Congest` *que resuelve* `DIÁMETRO` *en $\mathcal{O}(\sqrt{n})$ rondas si $|I| = \mathcal{O}(\sqrt{n})$ y $|K| = \mathcal{O}(n)$.*

Demostración. Para verificar que un grafo split es de diámetro 2 o 3, basta verificar si los nodos independientes están a distancia 2 o 3 entre ellos, luego basta verificar cada par de nodos independientes. Al haber $\sqrt{n}$ de estos vértices, cada uno se puede emparejar con $\sqrt{n}-1$ otros nodos, es decir, hay $\sqrt{n}\cdot(\sqrt{n}-1) \leq n$ combinaciones posibles, luego podemos hacer una asociación entre los pares posibles y los nodos del clique.

S.p.g. supongamos que los **id**'s de los nodos están en $[n + n^{\frac{1}{2}}]$ (si no basta que los nodos del clique comuniquen los **id**'s de los independientes, estos son a lo más $\mathcal{O}(\sqrt{n})$ mensajes), los nodos del clique conocen la identificación de sus vecinos y luego del proceso de eliminación de twins, conocen quienes son del clique y quienes son del independiente. Los id's que faltan

serán claramente de los independientes, de lo que se podrá saber cuales son las combinaciones posibles de estos, permitiendo así asignarlas de menor a mayor a los nodos del clique.

Una vez el nodo $v \in K$ tenga asociado al par $i, j \in I$, si existe un nodo $u \in K$ tal que $ui, uj \in E$, este le notificará a v. Si al menos un nodo le notifica a v, significa que su par si está a distancia 2. Si nadie lo hace, significa que están a distancia 3 y en ese caso el diámetro del grafo es 3, lo que es comunicado a todos los nodos. Si no existe tal mensaje, el diámetro del grafo es 2.

Es importante notar que está demostración está hecha para el caso en que $|I| \leq \sqrt{n}$, no para cuando $|I| = \mathcal{O}(\sqrt{n})$. En el caso más general, es necesario que cada nodo revise un número constante de parejas de nodos independientes. Este proceso no afecta la complejidad del protocolo. □

Este primer caso, donde el clique se hace cargo de los cálculos, abre la puerta al desarrollo de una idea importante, en la que el trabajo principal es realizado por el clique, mientras que el resto de los vértices anclados a esta estructura solo transmiten su información.

2.2.4. Diámetro de clique balanceado

En esta subsección será presentado un algoritmo para resolver el problema del diámetro en el caso de grafos split balanceados. Un grafo Split se dice balanceado si el número de nodos del clique $|K|$, al igual que el número de nodos del conjunto independiente $|I|$, es $\Theta(n)$. En otras palabras ambos conjuntos tienen aproximadamente la misma cantidad de nodos.

En el último resultado se estudió el caso en que el clique tiene un tamaño mucho mayor al conjunto independiente. En ese caso es posible adjudicar el trabajo de chequeo al clique, pues cada nodo del clique se encarga de revisar la conexión entre un par de nodos independientes. En el caso balanceado no será posible debido al número de nodos, porque cada uno de los nodos del clique tendría que revisar muchos posibles pares de nodos independientes (n para ser preciso). Al ser un grafo balanceado podemos hacer otro tipo de asociación, donde cada nodo del clique se encargue idealmente de un solo nodo independiente. Se verá que no es posible hacer una asociación 1 a 1, pero si asignar un conjunto de tamaño constante con alta probabilidad.

El proceso es entonces, asignar a cada nodo del clique un conjunto independiente, de manera que este nodo desempeñe su propio papel y a la vez simule el de los nodos asignados. ¿Por qué esto podría ser más eficiente si recién se vio que el número de combinaciones entre independientes sería muy grande? Pues podemos simular un modelo diferente al `Congest`, el modelo `Congested Clique`, donde todos los nodos se pueden comunicar entre sí, sin importar las conexiones. El modelo `Congested Clique` es un modelo sumamente poderoso, tanto que no se conocen técnicas que puedan probar cotas inferiores no triviales en este modelo [6, 21].

Al estar todos los nodos del clique conectados entre sí, si se les entrega toda la información

necesaria y sin sobrecargar a ninguno, se podría desarrollar un protocolo simulando el trabajo en el modelo `Congested Clique`.

En el modelo `Congested Clique`, el cálculo del diámetro se puede hacer de manera muy eficiente, ya que se puede asociar directamente a multiplicación de matrices. A cada grafo se le puede asociar una matriz de adyacencia, esta es una matriz de $n \times n$ en la que en la posición i, j hay un 1 si los nodos i y j están conectados y un 0 en caso contrario.

Teorema 2.2 *Sea G un grafo split balanceado, donde K el clique máximo de G e I es el conjunto de vértices independientes que cumple que $K \cup I = V$. Existe un protocolo probabilista en el modelo `Congest` que determina DIÁMETRO en $\mathcal{O}(n^{0.158})$ rondas con una probabilidad de $\frac{2}{3}$.*

Demostración. El procedimiento se divide en 4 etapas.

Primera etapa: La Primera etapa será aplicar el protocolo de la Proposición 2.5, de esta manera se eliminarán los twins y cada nodo sabrá en que conjunto se encuentra.

Segunda etapa: Identificar nodos del independiente con nodos del clique. Los nodos del clique escogen un orden entre 1 y n entre ellos, al conocer el **id** de todos basta con hacerlo de menor a mayor. Para identificar los nodos del independiente con los del clique basta reducir el espacio donde están a uno de tamaño n, una manera directa de hacerlo es tomar el **id** del nodo `mod` n.

Tercera etapa: Cada nodo v_i del clique que esté conectado a un nodo u independiente asociado al nodo v_j según la identificación anterior, le notifica a v_j que es vecino de u. De esta manera el nodo v_j recibirá todas las notificaciones del clique reconstruyendo así la vecindad del nodo u.

Cuarta etapa: Ahora que la información de cada nodo independiente esta contenida en un nodo del clique, es posible que cada nodo del clique simule su parte y la de $\mathcal{O}(1)$ nodos más con alta probabilidad en el modelo `Congested Clique`. Luego basta simular el protocolo para APSP (*all pairs shorthest path*) desarrollado en [22] para el caso en el que es un grafo no dirigido y sin pesos, donde cada nodo comparte su infromación y la de los nodos de los que está encargado y realiza sus computaciones.

De esta manera el procedimiento se ve de la siguiente manera:

Protocolo para DIÁMETRO

1. Ejecutar protocolo para la identificación de twins.
2. Ejecutar protocolo para identificar el clique K y el conjunto independiente I.
3. Nodos en K coordinan identificaciones entre $\{1, ..., n\}$.
4. Para cada $v \in I$, toman como nuevo identificador **id** `mod` n
5. Para cada $v \in K$, por cada $u \in N(v)$ tal que el nuevo identificador de u es j, v le comunica al nodo $j \in K$ que es vecino de u.
6. Los nodos de K simulan el protocolo para APSP del modelo `Congested Clique`

jugando su rol y el de los nodos que tienen asignados.

7. Comunican a los nodos de I el diámetro del grafo.

Veamos que este procedimiento es correcto. La primera etapa se ejecuta sin problemas en base a lo demostrado en los lemas 2.5 y 2.7. En cuanto a la segunda etapa, la parte que corresponde al clique se ejecuta como es descrito, pero la asociación entre estos y los nodos independiente puede tener ciertas complicaciones.

Lo más importante es verificar cuantos nodos independientes serán asignados a cada nodo del clique y con que probabilidad este es un número grande. Denotemos por $n(v)$ la variable aleatoria correspondiente al número de nodos asignados al nodo v. Ahora la probabilidad de que $n(v) = k$, para k cierto número natural, es la probabilidad de que k **id**'s sean igual a v `mod` n y el resto sea cualquier otro, es decir:

$$\mathbb{P}(n(v) = k) = \left(\frac{1}{n}\right)^k \left(\frac{n-1}{n}\right)^{n-k} = (n-1)^{n-k} \left(\frac{1}{n}\right)^n$$

Luego el número esperado de nodos que se le asigna a v es

$$\begin{aligned}
\mathbb{E}(n(v)) &= \sum_{k=0}^{n} k(n-1)^{n-k} \left(\frac{1}{n}\right)^n \\
&= \left(\frac{1}{n}\right)^n \frac{n(n-1^n) - (n-1)^n - n^2 + n + 1}{(n-2)^2} \\
&= \left(\frac{n}{(n-2)^2}\right)\left(\frac{n-1}{n}\right)^n + \left(\frac{1}{(n-2)^2}\right)\left(\frac{n-1}{n}\right)^n + \left(\frac{-n^2+n-1}{(n-2)^2}\right)\left(\frac{1}{n}\right)^n
\end{aligned} \tag{2.1}$$

Notemos que el término $\left(\frac{n-1}{n}\right)^n \leq 1$, esto junto a que $\left(\frac{n}{(n-2)^2}\right) \in \mathcal{O}(\frac{1}{n})$ y $\left(\frac{1}{(n-2)^2}\right) \in \mathcal{O}(\frac{1}{n^2})$ nos da que los dos primeros términos de la expresión son $\mathcal{O}(\frac{1}{n})$. Además $\left(\frac{-n^2+n-1}{(n-2)^2}\right) \in \mathcal{O}(1)$ por lo que el último término decrece mucho más rápido que $\frac{1}{n}$, luego $\mathbb{E}(n(v)) \in \mathcal{O}(\frac{1}{n})$. Por lo tanto existe $c_1 \in \mathbb{R}$ tal que $\mathbb{E}(n(v)) \leq c_1 \cdot \frac{1}{n}$.

Teniendo esto listo, podemos proseguir con estudiar la probabilidad de que algún nodo reciba muchos nodos independientes, pues este sería el caso en que el tiempo de computación escale. Sea $c_2 \in \mathbb{R}$ constante y usando la desigualdad de Markov tenemos que:

$$\mathbb{P}(n(v) \geq c_2) \leq \frac{\mathbb{E}(n(v))}{c_2} \leq \frac{c_1}{c_2 n} = \frac{c_3}{n}$$

Luego la probabilidad de que exista un nodo que tenga asignados más de c_2 nodos es:

$$\mathbb{P}(\exists v \in K : n(v) \geq c_2) \leq c_3$$

Escogiendo de manera adecuada las constantes, podemos designar $c_3 = \frac{1}{3}$. Luego tenemos un algoritmo probabilista que con una probabilidad de $\frac{2}{3}$ entrega a lo más un número constante de nodos independientes a cada nodo del clique.

Al final de la tercera etapa cada nodo conoce toda la información necesaria, puesto que el emparejamiento entre nodos del clique y nodos independientes ya está hecho y solo se precisa una ronda para transmitir las conexiones. Como además es un clique, cada nodo puede enviar directamente el mensaje.

En cuanto a la cuarta etapa, al estar cada nodo encargado de un número constante de nodos con una probabilidad de $\frac{2}{3}$, son capaces de simular un protocolo del modelo `Congested clique` en un mayor número de rondas. La correctitud del protocolo para el cálculo de los caminos más cortos entre cualquier par de nodos viene de [22].

En cuanto a la complejidad de este algoritmo, ya vimos que la primera etapa se puede hacer en un número constante de rondas. La segunda y tercera etapa por su parte solo requieren una ronda de comunicación cada una. Finalmente en la cuarta etapa, si bien la capacidad de computación de cada nodo es ilimitada, deberán enviar más mensajes que solamente los suyos. Con una probabilidad de $\frac{2}{3}$ la cantidad de mensajes que tendrá que enviar cada nodo solo escalara por un factor constante, luego puede replicar el protocolo en el mismo número de rondas.

Por otro lado, el costo del algoritmo a simular está directamente conectado con qué tan eficientemente se puede calcular la multiplicación de matrices. $\omega < 2.3728639$ corresponde al exponente de multiplicación de matrices [23] y la cota señalada es la mejor encontrada hasta el momento, aunque se conjetura que esta podría llegar a ser acotada por 2. El costo de esta parte del protocolo es de $\mathcal{O}(n^{1-\frac{2}{\omega}})$, dando así un algoritmo que corre en $\mathcal{O}(n^{0.158})$ con una probabilidad de $\frac{2}{3}$. □

Lo más poderoso de este protocolo, es que la etapa que solucionaba el problema del diámetro es independiente del procedimiento anterior, es decir, se podrían aplicar protocolos similares a este en el que varía la última etapa acorde al problema que se enfrenta. En esencia cualquier problema podría ser transportado al modelo `Congested Clique` si se está trabajando en un grafo split.

2.3. Generalización de grafos split

2.3.1. Problemática y modelo

Como se vio en las secciones anteriores, el trabajo con un grafo Split puede tener un enfoque bastante particular y útil. Si se logra pasar la información de los nodos del independiente al clique, es posible usar todas las herramientas que dispone el modelo `Congested Clique` para resolver el problema que se esté enfrentando. El único impedimento es entonces el traspaso de la información de un lado al otro del grafo. Pero si el cuello de botella es el traspaso de información y no particularmente la estructura en la que se está trabajando, entonces es posible variar la estructura. No es necesario que sean nodos independientes anclados a un clique, a priori si fuese una estructura lo suficientemente liviana podría hacerse algo similar.

Cuando se habla de estructura liviana en general se hace referencia a un conjunto de nodos conectados entre ellos por pocas aristas, no hay una definición absoluta y clara para este concepto, pero en esta tesis se hará referencia a que $|E| = \mathcal{O}(n)$, a menos que se explicite algo distinto. Algunas estructuras de este estilo pueden ser grafos donde el grado máximo de los nodos está acotado ($\Delta \leq k$), árboles o grafos de degenerancia acotada. Estas son las estructuras con las que se trabajará en un inicio y con las que se extenderá la noción más básica de un grafo split.

Modelo: Formalizando esta idea, se tendrá un grafo con $\Theta(n)$ nodos, $\Theta(n)$ de ellos forman un clique, mientras que los otros $\mathcal{O}(n)$ forman una estructura diferente. Además los vértices de esta estructura cumplen que cada uno de ellos está conectado a al menos un nodo del clique (sin limite superior). Notar que no basta que hayan $\Theta(n)$ canales de comunicación pues se pueden dar problemas de congestión de información. Por ejemplo si un solo nodo tiene los $\Theta(n)$ canales de comunicación y es el único nodo de la estructura que se conecta con el clique se generará un cuello de botella en este nodo.

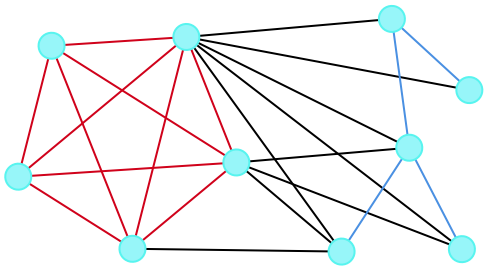

Figura 2.4: Un ejemplo de una generalización del grafo split con una familia hereditaria. En rojo marcado un clique de tamaño 5, conectado a nodos que forman un grafo planar en azul.

El problema más complicado que se puede atacar es el de reconstrucción, donde la idea es que al final del proceso cada nodo del grafo conozca todos los nodos y aristas del grafo. De manera más formal, definiremos dos versiones del problema de manera similar a como lo hacen en [10]. El primer problema es el *problema de reconstrucción fuerte* $\mathcal{G}$-`Strong-Rec`:

$\mathcal{G}$-Strong-Rec

Input: Un grafo arbitrario G.

Output:
$$\begin{cases} \text{todas las aristas de } G & \text{si } G \in \mathcal{G}; \\ \text{rechazar} & \text{en otro caso.} \end{cases}$$

Es decir que si la red acepta, todos los nodos deben saber si $G \in \mathcal{G}$ y además si esto es efectivo, también saber todas las aristas de G. Otra versión del problema es el *problema de reconstrucción débil* $\mathcal{G}$-`Weak-Rec`. Este es un problema con *promesa*, donde el input G tiene la promesa de ser de la clase $\mathcal{G}$.

$\mathcal{G}$-Weak-Rec

Input: $G \in \mathcal{G}$.
Output: todas las aristas de G.

En el caso de que una red resuelva $\mathcal{G}$-Strong-Rec y $G \in \mathcal{G}$ o resuelva $\mathcal{G}$-Weak-Rec cada nodo tendrá conocimiento de todas las conexiones y por lo tanto todos los nodos. Luego cada nodo es capaz de resolver cualquier otro problema, pues los nodos tienen una capacidad infinita de procesamiento.

En [10] se resuelve estos problemas en el modelo **BCAST** para familias hereditarias y otros casos. Las complejidades computacionales de esto se muestran en la Tabla 2.2 y en la Tabla 2.1.

Tabla 2.1: Complejidad para problemas de reconstrucción en el modelo **BCAST**.

Problema	**Rondas**	**Ancho de banda**
$\mathcal{G}$-Weak-Rec hereditario moneda privada	1	$\mathcal{O}(\log(\lvert\mathcal{G}_n\rvert)/n)$
$\mathcal{G}$-Weak-Rec moneda privada	1	$\mathcal{O}(\sqrt{\log(\lvert\mathcal{G}_n\rvert \log n)} + \log n)$
$\mathcal{G}$-Strong-Rec determinista	2	$\mathcal{O}(\sqrt{\log(\lvert\mathcal{G}_n\rvert \log n)} + \log n)$

Tabla 2.2: Complejidad para problemas de reconstrucción en el modelo Congested Clique.

Problema	**Rondas**	**Ancho de banda**
$\mathcal{G}$-Weak-Rec determinista	2	$\mathcal{O}(\log(\lvert\mathcal{G}_n\rvert)/n + \log n)$
$\mathcal{G}$-Strong-Rec determinista	3	$\mathcal{O}(\log(\lvert\mathcal{G}_n\rvert)/n + \log n)$
$\mathcal{G}$-Strong-Rec moneda privada	2	$\mathcal{O}(\log(\lvert\mathcal{G}_n\rvert)/n + \log n)$
$\mathcal{G}$-Strong-Rec moneda privada	1	$\mathcal{O}(\sqrt{\log(\lvert\mathcal{G}_n\rvert \log n)} + \log n)$

2.3.2. Reconstrucción de familias hereditarias

Se parte con algunas definiciones para trabajar detalladamente el problema. La clase de grafos en la que se trabajará se define a continuación.

Definición 2.1 *Sea $\mathcal{G}$ una familia de grafo hereditaria. Sea $\mathcal{J}(\mathcal{G})$ una clase de grafos definida como:*

$$\mathcal{J}(\mathcal{G}) = \Big\{ G \mid V = V_1 \cup V_2,\ G[V_1] = K_n,\ |V_2| = \Theta(n), G[V_2] \in \mathcal{G},\ \forall u \in V_2\ \exists v \in V_1 : uv \in E \Big\}$$

En palabras, se tendrá un clique de tamaño n aproximadamente y una estructura asociada a una clase $\mathcal{G}$ de un tamaño similar, que cumple que cada nodo debe estar conectado a al menos un nodo del clique. Durante el resto del trabajo normalmente nos referiremos a $G[V_1]$ como K y a $G[V_2]$ como H.

Uno de los principales resultados demostrados en [10] muestra que en el modelo **BCAST** es posible reconstruir un grafo G si pertenece a una familia hereditaria $\mathcal{G}$ en $\mathcal{O}(\log\big(|\mathcal{G}_|\big)/n)$ con alta probabilidad. Este resultado será adaptado al modelo `Congest` y a la familia de grafos $\mathcal{J}$.

Proposición 2.8 ([10]) *Sea $\mathcal{G}$ una familia hereditaria de grafos. Existe un protocolo de una ronda en el modelo **BCAST** con moneda privada que resuelve $\mathcal{G}$-`Strong-Rec` con alta probabilidad y ancho de banda $\mathcal{O}(\log(|\mathcal{G}_n|)/n)$.*

Supongamos tenemos un grafo $G \in \mathcal{J}(\mathcal{G})$, donde cada nodo v del clique K tiene conocimiento de un conjunto de nodos de H y sus vecindades. El nodo v podría representarse a sí mismo y al conjunto asignado para ejecutar un protocolo del modelo `Congested Clique`. Este proceso como se vio en el caso de un grafo split explota el poder del modelo `Congested Clique` siendo que se está trabajando en el modelo `Congested`. Luego lo que faltaría es una manera de entregar la información del conjunto asignado al nodo v.

Teorema 2.3 *Sea $\mathcal{G}$ una familia hereditaria de grafos y sea $G \in \mathcal{J}(\mathcal{G})$. Además sea $H \in \mathcal{G}$ el subgrafo correspondiente a G. Sea $\mathcal{P}$ un problema de decisión para el grafo H que admite un protocolo $\mathcal{A}$ en el modelo `Congested Clique` en $f(n)$ rondas.*

Entonces el problema $\mathcal{P}$ en G admite un protocolo en el modelo `Congest` en $\mathcal{O}(\frac{\log|\mathcal{G}_n|}{n\log n} + f(n))$ rondas con alta probabilidad en, si se conoce quienes son los nodos de K y de H.

Demostración. La demostración consistirá en mostrar un algoritmo que realiza esencialmente 3 pasos. Durante el algoritmo se usa un número primo p que se especifica en el algoritmo. Además los nodos del clique escogerán un nuevo **id** en $[n]$. Es importante destacar que como ya no nos encontramos en un grafo split, la identificación del clique y el resto del grafo se hacen más complejas, es por esto que se pide en el teorema como hipotesis.

Lo primero es adaptar el protocolo que se muestra en [10] para los nodos de H. Se inicia con que cada nodo calcule su fingerprint (proceso realizado en [10]) y lo comunica al nodo del clique que tenga asignado. Esta asociación se hace de manera análoga al proceso hecho para el grafo split, es deicr, toma su **id** `mod` n.

En segundo lugar los nodos de K reconstruirán al grafo H de manera de conocer no sólo a los nodos que tienen asignados si no que también sus vecindades. Esta reconstrucción se hace compartiendo los fingerprints y buscando un grafo que tenga el mismo fingerprint en la familia $\mathcal{G}$ acorde a lo discutido en [10]. De esta manera cada nodo de K conocerá las vecindades de los nodos que tiene asignados.

Finalmente, como tercera etapa el clique K simulará el algoritmo correspondiente para resolver el problema P. Esto lo realiza haciendo como si estuviese en el modelo `Congested Clique` y cada nodo jugando el papel suyo y de sus nodos asignados.

Así el algoritmo queda de la siguiente manera:

Protocolo reconstrucción

1. Cada nodo i en H computa p, el menor primo mayor a $n^3 \cdot e \cdot 2^{(g(n)/n)}$, donde $g(n) = n \cdot \max_{k \in [n]} \frac{\log |\mathcal{G}_k|}{k}$.
2. Cada nodo i en H escoge $T_i \in \mathbb{F}_p$ (donde $\mathbb{F}_p$ es el cuerpo de p elementos) uniformemente al azar usando una moneda privada.
3. Cada nodo i en H computa $FP(x_i, T_i)$.
4. Los nodos de K se ordenan entre 1 y n.
5. Cada nodo i en H comunica $FP(x_i, T_i)$ a algún nodo del clique
6. Cada nodo del clique le transmite el mensaje recibido al nodo correspondiente al **id** `mod` n del nodo de H que lo envió.
7. Los nodos de K comparten los fingerprints entre ellos.
8. Cada nodo del clique construye $T = (t_1, ..., t_n)$ y $FP(H, T) = (FP(x_1, t_1), ..., FP(x_n, t_n))$
9. Los nodos de K buscan H' tal que $FP(H, T) = FP(H', T)$. De esta manera los nodos de K reconstruyen H.
10. Los nodos de K simulan el protocolo para el problema $\mathcal{P}$ haciendo su papel y el del conjunto de nodos asignados como si estuvieran en el modelo `Congested Clique`.
11. Los nodos de H aceptan. Los nodos de K aceptan si ellos y el conjunto que representan acepta. Si al menos un nodo que representan rechaza entonces rechazan.

Veamos ahora la correctitud de este proceso.

Completeness: Supongamos que el grafo G está configurado de manera tal que el problema $\mathcal{P}$ es satisfecho, es decir, los nodos debiesen aceptar. Esto significa que al momento de detenerse el protocolo $\mathcal{A}$, si estuviésemos trabajando en el modelo `Congested Clique` todos los nodos aceptarían. Como cada nodo del clique simula este proceso haciendo su papel y el de un conjunto de nodos del grafo H (cubriendo a todos los nodos de H), al detenerse el algoritmo todos los nodos deberían aceptar. En vista de que este proceso es realizado sólo por los nodos de K, todos estos nodos aceptan, además por defecto los nodos de H aceptan.

Soundness: Supongamos que el grafo G está configurado de manera tal que el problema $\mathcal{P}$ no es satisfecho, es decir, al menos un nodo debiese rechazar. Argumentando de manera análoga la punto anterior, el protocolo $\mathcal{A}$ debiese entregar al menos un nodo que rechace al ser simulado por K. El nodo que rechaza, será un nodo del clique o alguno representado por un nodo del clique, luego el nodo correspondiente del clique rechazará.

Es importante hacer notar que cada nodo de H será representado por alguien en el clique. Y que además los nodos del clique efectivamente tendrán la información necesaria para re-

presentar al conjunto asignado. Esto viene de que el algoritmo de reconstrucción es correcto, demostrado en [10].

Finalmente es necesario analizar la complejidad de este protocolo. La información transmitida, los fingerprints, son de tamaño $\mathcal{O}(\log |\mathcal{G}_n|/n)$. Estos mensajes son enviados desde H al clique y luego desde cada nodo del clique al nodo correspondiente. Por lo demostrado en la Proposición 2.2, tendremos que con alta probabilidad a cada nodo del clique se le asignará un conjunto de tamaño constante. Luego, si un nodo del clique recibe muchos mensajes de distintos nodos, tendrá que transmitirlos a diferentes nodos del clique, por lo que no requerirá transmitir más que $\mathcal{O}(\log |\mathcal{G}_n|/n)$ bits. De la misma manera, la complejidad no aumentará al comunicar estos mensajes entre el clique en la siguiente ronda. Esto sumado a qué cada nodo puede transmitir hasta $\mathcal{O}(\log n)$ bits por cada canal da que la complejidad de la primera y segunda parte del proceso es de $\mathcal{O}(\frac{\log |\mathcal{G}_n|}{n \log n})$.

Luego de la reconstrucción, los nodos del clique deben simular el protocolo $\mathcal{A}$, jugando su papel y el de los nodos asignados, el cuál con alta probabilidad es constante, luego en cada ronda deberán mandar a lo más un número constante de veces la información que harían normalmente. Por lo tanto la complejidad de este proceso seguirá siendo de $f(n)$. Juntando todo obtenemos que la complejidad del proceso es $\mathcal{O}(\frac{\log |\mathcal{G}_n|}{n \log n} + f(n))$. □

En la Tabla 2.3 y en la Tabla 2.4 se pueden apreciar problemas y sus respectivas complejidades en el modelo `Congested Clique`. Estos son ejemplos de problemas que podrían ser resueltos en grafos de la familia $\mathcal{J}(\mathcal{G})$, con $\mathcal{G}$ una familia hereditaria, a través del protocolo descrito en el Teorema 2.3.

Tabla 2.3: Complejidad para diferentes problemas en el modelo Congested Clique.

Problema	**Complejidad**	**Referencia**
Triangle and 4-cycle counting	$\mathcal{O}(n^{1-\frac{2}{\omega}})$	[24]
1+o(1) APSP weighted directed approximation	$\mathcal{O}(n^{1-\frac{2}{\omega}})$	[24]
2+o(1) APSP weighted directed approximation	$\tilde{\mathcal{O}}(n^{1/2})$	[25]
2+o(1) APSP unweighted undirected approximation	$\tilde{\mathcal{O}}(n^{1/2})$	[25]
Girth	$\mathcal{O}(n^{1-\frac{2}{\omega}})$	[24]
4-cycle detection	$\mathcal{O}(1)$	[24]
k-cycle detection	$2^{\mathcal{O}(k)}n^{1-\frac{2}{\omega}}$	[24]
Weighted directed diameter U	$\mathcal{O}(Un^{1-\frac{2}{\omega}})$	[24]
APSP unweighted undirected	$\mathcal{O}(n^{1-\frac{2}{\omega}})$	[24]
MST random	$\mathcal{O}(1)$	[26]
MST deterministic	$\mathcal{O}(\log \log n)$	[27]
Computing size of max matching	$\mathcal{O}(n^{1-\frac{2}{\omega}} \log n)$	[28]
Computing edges in a perfect matching	$\mathcal{O}(n^{1-\frac{1}{\omega}})$	[28]
Gallai-Edmonds descomposition	$\mathcal{O}(n^{1-\frac{1}{\omega}})$	[28]
Min vertex cover in bipartite graphs	$\mathcal{O}(n^{1-\frac{1}{\omega}})$	[28]
Min Cut 1+$\mathcal{O}(1)$ aprroximation	$\mathcal{O}(1)$	[21]
Min Cut exact	$\mathcal{O}(\log^3 n)$	[21]
Min Cut exact	$\mathcal{O}(\log^2 n)$	[21]
Routing Problem	$\mathcal{O}(1)$	[29]
Sorting Problem	$\mathcal{O}(1)$	[29]
$(\Delta+1)$-list-coloring	$\mathcal{O}(1)$	[30]
$(\Delta+1)$-coloring	$\mathcal{O}(1)$	[30]

Tabla 2.4: Complejidad para encontrar una aproximación del árbol de Steiner en el modelo Congested Clique.

Problema	**Complejidad**		**Referencia**
	Rondas	**Mensajes**	
Steiner Tree approximation	$\tilde{\mathcal{O}}(n^{1/3})$	$\tilde{\mathcal{O}}(n^{7/3})$	[31]
Steiner Tree approximation	$\mathcal{O}(S+\log \log n)$	$\mathcal{O}(S(n-t)^2+n^2)$	[31]

Capítulo 3

Modelo de Iteraciones Dinámicas

3.1. Definición del modelo

Hasta ahora se ha trabajado principalmente con PLSs y 2 modelos de computación distribuida, el modelo `Congest` y el modelo `Congested Clique`. La flexibilidad del concepto de computación distribuida y las incontables posibilidades de problemas y situaciones a las que se puede enfrentar, conllevan a que la cantidad de modelos en los que se pueda trabajar sea virtualmente infinita.

El trabajo realizado en esta tesis ha presentado situaciones en los que dado un grafo G de n nodos, es necesario resolver cierto problema en la configuración correspondiente. En ese sentido son modelos estáticos en los que los conjuntos de aristas y vértices no varían. Pero nada impide visualizar un modelo en que esta situación no sea así.

Se han desarrollado modelos en los que pueden existir cambios en la topología del grafo, donde las aristas (vistas como canales de comunicación) pueden caer y volver a estar en servicio [32]. Problemas en este tipo de modelo han sido ampliamente estudiados [33, 34].

El principal objetivo de esta sección es presentar y desarrollar un nuevo modelo de computación distribuida al que se denomina `Modelo de Iteraciones Dinámicas` (MID). Este se caracteriza por ser un modelo en que el grafo evoluciona dinámicamente, donde en cada etapa un nuevo nodo y sus respectivas aristas son agregados. Se comienza definiendo el modelo para luego presentar y resolver diferentes problemas, comparando como se comporta éste con respecto a otros modelos.

3.1.1. MID

El MID es un modelo de computación distribuida. Este modelo asume una red modelada como un grafo $G = (V, E)$ conexo. Cada nodo $u \in V$ es dotado con un identificador $\mathbf{id}(u)$ de $\mathcal{O}(\log n)$ bits de largo tales que todos los identificadores son distintos. Los nodos son honestos, es decir, la información que entregan es confiable, y las conexiones son seguras (i.e. el modelo es libre de fallas).

Los nodos son iniciados de manera secuencial. En una primera etapa se cuenta con un único nodo disponible. Supongamos que en la etapa j se cuenta con un grafo $G' = (V', E')$. En la etapa $j+1$ se revelará un nodo u y las aristas de $G[V' \cup u]$, es decir, el nodo u y todas sus aristas en G asociadas a los nodos que ya han sido revelados. La elección del siguiente nodo a ser revelado la hace un adversario. En cada etapa es revelado un solo vértice y sus respectivas aristas, en particular, en la etapa j se contará con j nodos del grafo G y todas las aristas de G que conecten a estos j nodos. Si en una etapa i se cuenta con el grafo G' revelado, el siguiente nodo a revelar u debe estar en $N(G')$, así el grafo revelado es conexo en todo momento.

En cada etapa los nodos revelados se inician al mismo tiempo y se procede en una secuencia sincrónica de rondas. En cada ronda todos los vértices envían mensajes a sus vecinos en G' (sus vecinos revelados), reciben mensajes de sus vecinos y realizan cálculos o algoritmos locales individuales. Los mensajes enviados en una misma ronda pueden ser diferentes. No hay límite en el poder de computación de cada nodo. Sin embargo, similar al modelo CONGEST las conexiones están sujetas a una restricción: en cada ronda, no más de $\mathcal{O}(\log n)$ bits pueden atravesar cada canal. Un nodo al ser revelado solo conoce los **id**'s de sus vecinos ya revelados. Los nodos son capaces de guardar información de etapa en etapa y al ser revelado un nodo solo se enteran sus vecinos.

En cada ronda del algoritmo, cada uno de los n nodos tiene que decidir si para o continúa. Un algoritmo se detiene cuando todos los nodos paran. La medida de complejidad es el número de rondas sincrónicas requeridas para que el algoritmo se detenga en una etapa cualquiera. Es decir, si se trabaja un problema $\mathcal{P}$ de decisión, un algoritmo $\mathcal{A}$ lo resuelve si es capaz de resolverlo en cada etapa y su complejidad será el máximo número de rondas que le toma entre todas las etapas.

3.2. Problemas

En esta sección se analizan diferentes problemas aplicados al MID, los primeros son problemas que se pueden resolver de manera directa o muy simple y que entregan un algoritmo muy eficiente, para así visualizar el poder del modelo. Luego se presenta un problema en el que la cota inferior es grande, demostrando así que el modelo no es todopoderoso. Finalmente se presentan modelos similares y problemas resueltos en estos modelos para ver como se comparan.

3.2.1. Problemas directos

Notese que este es un modelo que actualiza constantemente la información de los nodos. Además la información de la red no se pierde entre rondas. Luego, es posible que los nodos guarden información importante que permita realizar la actualización (al llegar un nuevo nodo) de manera rápida. Por ejemplo, si se ve un problema local, al llegar un nuevo nodo basta con que revise la información guardada por sus vecinos para tomar una decisión. Después

de esto procede a guardar la información necesaria para repetir el proceso. Por lo mismo, en este modelo, los problemas locales serán usualmente más rápidos de resolver. Algunos de estos problemas son los siguientes:

- LÍDER $= \Big\{(G, I) | I : V(G) \to \{0, 1\}$ y $|v \in V(G) : I(v) = 1| = 1\Big\}$
- MIS $= \Big\{(G, I) | I : V(G) \to \{0, 1\}$ y $\{v \in V(G) : I(v) = 1\}$ forman un conjunto independiente maximal$\Big\}$
- MATCHING MAXIMAL $= \Big\{(G, I) | I : V(G) \to \{0, 1\}$ y $\{e = uv \in E(G) : I(v) = I(u) = 1\}$ forman un conjunto matching maximal$\Big\}$
- Δ-FREE $= \Big\{(G, I) |$ tal que K_3 no es subgrafo de $G\Big\}$

Veamos ahora ideas de como resolver estos distintos problemas en MID.

LÍDER: Es posible escoger al nodo inicial como el líder. Luego para el resto de los vértices solo se les comunica etapa a etapa quién es el líder. Este proceso claramente escoge un líder y no permite que se escoja uno adicional. Al final del proceso todos los nodos conocen el **id** del líder.

MIS: En el problema de MIS, al estar buscando un conjunto maximal, posibilita una revisión local de la independencia. Luego, similar al proceso anterior, el primer nodo es seleccionado para ser parte del conjunto independiente I. Cada nodo v que es revelado solo debe verificar si tiene como vecino un nodo que es parte del conjunto independiente I o no. Si no hay ningún vecino del independiente, entonces v se considera parte de I, en caso contrario no lo hace. Al final del proceso se tendrá un conjunto independiente, puesto que se agregan nodos cuyos vecinos no lo son. Además este conjunto independiente será maximal porque si un nodo se puede agregar, es agregado.

MATCHING MAXIMAL: Para encontrar un matching maximal se procede de manera parecida al caso del conjunto independiente. Por defecto los nodos declaran que no son parte del matching, excluyendo al primer nodo. Cada vez que llega un nodo nuevo u verifica en su vecindad si existe algún nodo tal que no es parte del match. Si tiene vecinos que no son parte del matching, selecciona al nodo v de menor **id** de su vecindad y le comunica que la arista que los conecta forma parte del matching. Con esto tanto u como v se declaran parte del match. Nuevamente, en cada paso se mantiene un matching y además todo par de vértices que puede ser agregado, es agregado.

Δ-FREE: Finalmente en el problema de verificar que el grafo no tiene triángulos (C_3) como subgrafo, la idea será verificar en cada paso si se forma un triangulo. Lo que se usa es que se inicia con un grafo sin triángulos. Se tendrá entonces que en cada iteración, al revelarse un nuevo nodo v, su vecindad le avisa a sus propios vecinos que ven al nuevo nodo. Si a ningún vecino de v llega un mensaje de notificación, significa que el nuevo nodo no ha

formado un triangulo y se mantiene la invariante, por lo que todos aceptan. En cambio si dos vecinos de v son vecinos, formando un ciclo de largo 3, rechazarán.

Lo que permite revisar los triángulos es su localidad, pero ¿es tan fundamental que tenga diámetro 1 para chequear esta estructura? ¿Qué tan complicado puede volverse si expandimos levemente la topología? Son estas preguntas las que empujan la siguiente sección, donde se estudian estructuras de 4 nodos.

Se puede ver cómo hay ciertas estructuras, que son fáciles de verificar, con las mismas ideas mencionadas antes pueden estudiarse grafos completos, independientes e incluso árboles, pero la simple combinación de algunas de estas las convierte en tareas mucho menos intuitivas. Un ejemplo es la verificación de un grafo split, el cual no es más que un clique junto a un independiente. Este problema es complicado principalmente porque el dinamismo del modelo permite que las estructuras principales del grafo varíen. Luego, es más difícil mantener una invariante como en los problemas anteriores. Para ser más precisos, el clique que se considere en alguna etapa puede no ser el clique final de la estructura. Para ver por qué esto es un problema podemos imaginar la siguiente secuencia de etapas (ver Figura 3.1):

1. Un nodo v es rebelado conectado a un K_4 de manera completa, lo que forma un K_5.
2. Un nodo u es rebelado, también conectado de manera completa al K_4.
3. Un nodo w es rebelado, conectado al K_5 formado por el nodo u.

Por lo tanto entre etapas el clique del grafo split puede variar, lo que conlleva a que sea necesario identificar nuevamente todo el grafo.

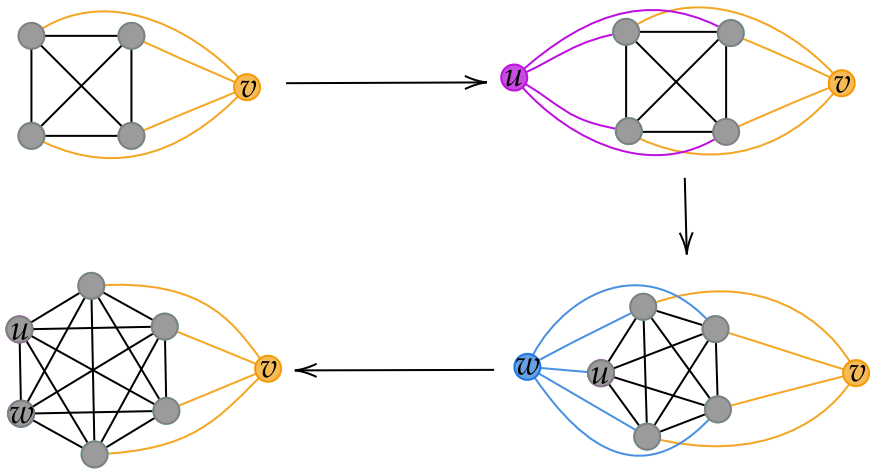

Figura 3.1: Una secuencia de etapas en las que el clique de un grafo split varía.

3.2.2. Subgrafos conexos de 4 nodos

A medida que el tamaño de los subgrafos (y más específicamente su diámetro) aumenta, las verificaciones se hacen cada vez menos locales, evitando que la respuesta sea simple. En

el caso de los subgrafos de cuatro vértices podemos identificar algunos de manera más simple que otros. Además dependiendo de si buscamos un subgrafo cualquiera o uno inducido, la complejidad varía. Esta subsección se centrará en el caso inducido, que es el más complicado. El problema central será entonces:

- **H**-FREENESS $= \Big\{(G, I)|\text{tal que } G \text{ tiene una copia isomorfa de } H \text{ como subgrafo}\Big\}$

  donde los grafos que se prohíben son los grafos de cuatro nodos conexos.

Para los grafos de cuatro vértices inducidos hay dos estructuras fundamentales que los forman a todos, a través de diferentes combinaciones. Estas estructuras son: triángulos y caminos de largo 3, ver Figura 3.2. Los triángulos son los ciclos de largo 3, o lo que es igual, un grafo completo de tres nodos. Los caminos de largo 3 los denotaremos como CH y son equivalentes a los grafos bipartitos completos de 3 nodos. Lo importante de estas estructuras es que son locales, algo que se ha visto que es un gran aporte en este modelo.

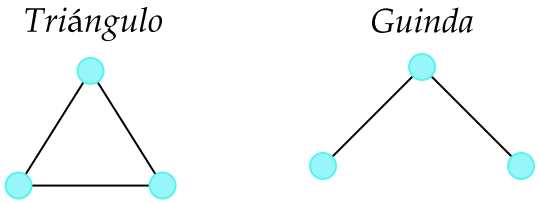

Figura 3.2: A la izquierda un *triángulo* o C_3, a la derecha CH o P_3

Se demostrará más adelante que si cada nodo conoce tanto los triángulos como los CHs en los que se encuentra basta para resolver **H**-FREENESS si **H** es un grafo conexo de cuatro nodos. El problema será que actualizar esta información toma muchas rondas, pues cada nodo que se rebele puede encontrarse en muchos triángulos. Para ser precisos, cada nodo v puede tener n vecinos, cada pareja de vecinos junto a v pueden formar un triangulo, luego la cota superior de triángulos en los que puede estar v es de n^2. Esto es mucha información, pero es posible comunicarla en n rondas. Es importante notar que las parejas de vecinos con los que no forma un triangulo, forman un CH, luego basta conocer una estructura para identificar la otra.

Lema 3.1 *Sea G un grafo. Al agregarle un nuevo nodo v, es decir, al pasar a una siguiente etapa en el MID, es posible comunicarle los triángulos y CHs en los que se encuentra a v en $\mathcal{O}(n)$ rondas.*

Demostración. Si bien v puede encontrarse en n^2 triángulos, visto desde la perspectiva de un vecino u, este puede formar solo $n-1$ triángulos con v. El algoritmo será:

Protocolo reconocimiento de triángulos y CHs

1. Los vecinos de v notifican que son vecinos del nuevo nodo.

2. Cada nodo u verifica que triángulos forma con v dependiendo de si recibe una

notificación de algún vecino.
3. Cada nodo u notifica a v el triángulo que forma con otro nodo w, siempre cuando $\mathbf{id}(u) > \mathbf{id}(w)$.
4. El vértice v asigna a todas las otras combinaciones de vecinos como CHs (las que no son triángulos).
5. Cada vecino u asigna un CH con v y sus vecinos que no eran vecinos de v.

Analicemos la correctitud de este proceso. Sea u, w vecinos de v tales que forman un triangulo. en la primera etapa tanto u como w notificarán la aparición de v, luego sabrán que forman un triangulo. Sin perdida de la generalidad supongamos que $\mathbf{id}(u) > \mathbf{id}(w)$, luego en la segunda etapa u le notificará a v que forman un triangulo con w. En cuanto a la complejidad computacional, la primera etapa es solo una ronda, para la segunda, cada vecino de v puede formar a lo más $n-1$ triángulos con v, suponiendo que es el vértice de mayor **id**, tendrá que enviar a lo más $n-1$ mensajes. Luego el protocolo toma $\mathcal{O}(n)$ rondas.

Es importante notar que en el caso de los triángulos, todo nodo del triángulo sabe que forma parte de este. En cambio para CH solo el nodo central de la estructura lo sabrá.

□

Con la identificación de las estructuras básicas podemos seguir al siguiente resultado. En la Figura 3.3 se pueden apreciar los grafos de 4 nodos conexos.

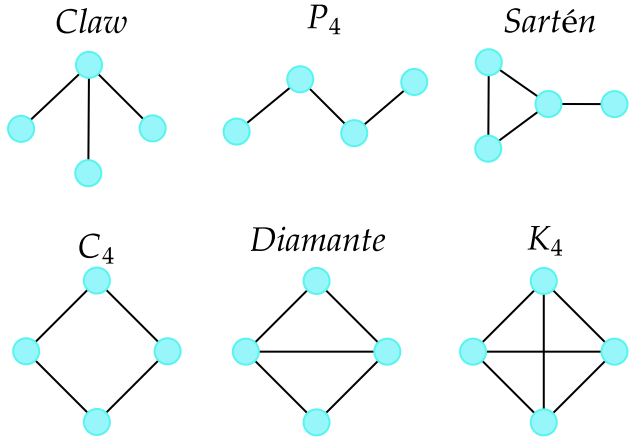

Figura 3.3: Todas las estructuras de 4 nodos conexas

Lema 3.2 *En el MID se puede resolver* ***H*-FREENESS** *para cualquier estructura de 4 nodos conexa en* $\mathcal{O}(n)$.

Demostración. El primer paso en cada etapa será que los nodos actualicen tanto los triángulos como CHs en las que se encuentran. Veremos el análisis que deben hacer los nodos para cada grafo de 4 nodos conexo, suponiendo que en la etapa anterior no se encontraba este como subgrafo inducido. Si la estructura es detectada por algún nodo, este rechazará, en caso

contrario aceptará. En un CH se llamará líder al nodo central. El líder además es quien está al tanto de la existencia de CH. A los otros dos nodos se les llamará hijos.

- $Claw = K_{1,3}$: Basta que cada nodo x verifique si hay un conjunto de 3 nodos con quienes forma tres CHs distintas.

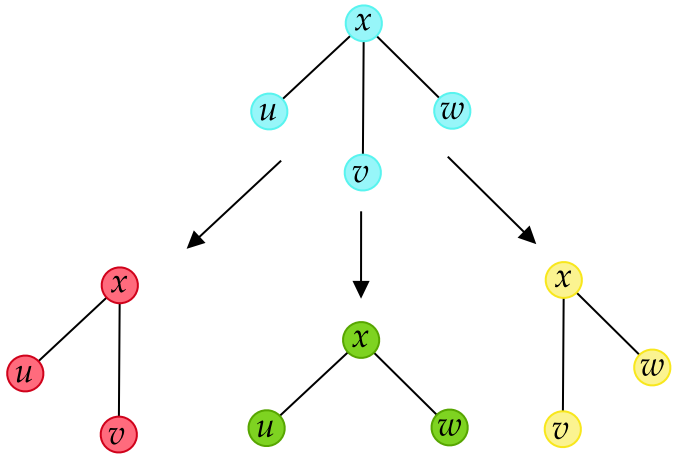

Figura 3.4: El grafo *claw* y los 3 CHs que lo conforman.

- P_4: Para que haya un camino de largo 4 deben haber dos CHs consecutivos, donde ambos líderes son además hijos del otro. Luego basta que los líderes le notifiquen a sus hijos que son el líder de un CH. Si para dos nodos esto es mutuo, entonces comparten quien es el otro de sus respectivos CHs. Luego, si no se forma un triángulo, entonces es un camino de largo 4.

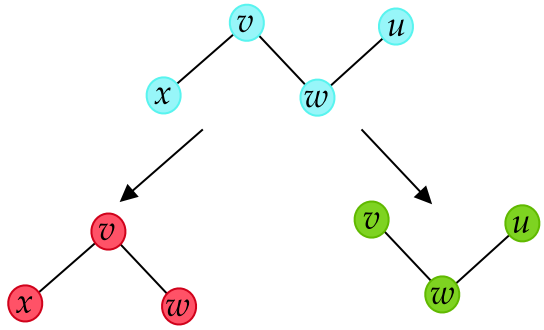

Figura 3.5: Un camino de largo 4. En este caso v es líder del CH formado por x, v y w, mientras que w es líder del CH formado por v, w y u. Además v es padre de w en el primer CH mientras que w es líder de v en el segundo, pero no forman un triángulo pues x y u son distintos.

- *Sartén = triángulo con una arista y vértice extra*: Esta estructura se puede analizar como un triangulo donde uno de los nodos, v forma CHs con los otros dos nodos. Además de esto es necesario que estos 2 CHs calcen en el nodo que no está en el triángulo. Luego basta que en cada triángulo los nodos verifiquen si ocurre esa situación.

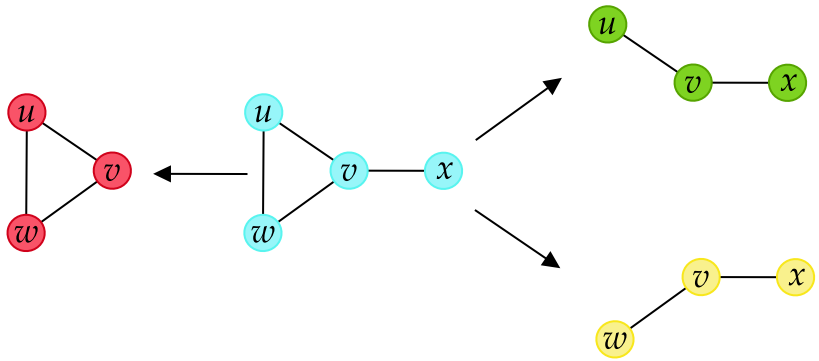

Figura 3.6: El grafo *sartén*, formado por un triángulo y dos CHs. EL factor común de las tres estructuras es v, por lo que por su cuenta es capaz de verificar la existencia de este subgrafo..

- C_4: si en el paso anterior no había un C_4, basta con que los nodos vecinos notifiquen la aparición del nodo v. Luego, si un líder de un CH recibe la notificación de ambos nodos de la estructura y a su vez no es vecino de v entonces se formó un C_4.

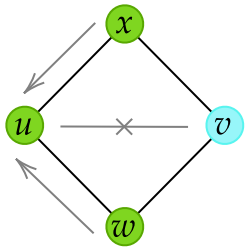

Figura 3.7: El grafo C_4. Una vez es revelado un nuevo nodo v, sus vecinos x y w le notifican esto a u que identifica la formación de un C_4.

- *Diamante = C_4 con una arista extra*: Es cuando hay exactamente dos triángulos consecutivos que comparten dos nodos. Luego, al aparecer un nuevo nodo, sus vecinos deben notificar esto. Si en un triángulo dos nodos notifican esto y no el tercero, significa que se formó un diamante.

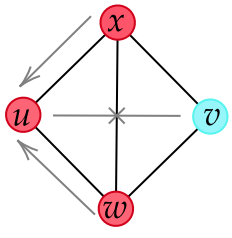

Figura 3.8: El grafo diamante. Este caso es análogo al de C_4, solo que la estructura que se tiene ahora es un triangulo en vez de un CH.

- K_4: corresponde a un triángulo donde todos sus vértices ven al nuevo nodo. Luego, al notificar los vecinos del nuevo nodo, todos los nodos del triángulo recibirán mensajes de sus dos vecinos.

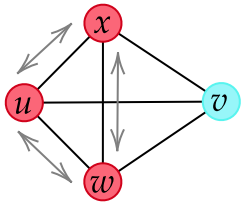

Figura 3.9: El grafo K_4. De manera similar a los últimos dos casos los vecinos del nuevo nodo v notifican su llegada, aquí, u, w y x se comunican entre todos .

□

Se puede apreciar como a medida que el núcleo del problema se aleja de lo local se vuelve más complicado, propiedad que explotaremos para demostrar una cota inferior en este modelo.

3.2.3. Cota inferior: Clique Máximo

En esta sección se presenta un importante problema, la detección del clique máximo en un grafo. Se demuestra que este problema no se puede resolver de manera más eficiente que lineal, demostrando así el amplio rango de problemas que presenta este modelo.

$$\text{MAX CLIQUE} = \Big\{(G, I) | I : V(G) \to \{0, 1\} \text{ y } \{v \in V(G) : I(v) = 1\} \text{ forman un clique máximo}\Big\}$$

Se mostrará entonces que es posible resolverlo en tiempo lineal y luego se verá que esto es lo mejor que se puede hacer.

Lema 3.3 *Existe un protocolo para resolver* MAX CLIQUE *en* $\mathcal{O}(n)$ *rondas en el* MID.

Demostración. El certificado que se le entregará a cada vértice será que cada nodo conozca los triángulos en los que se encuentra y el tamaño del clique máximo, en otras palabras, cada nodo conoce las conexiones entre sus vecinos. Siendo este el certificado, basta con replicar el Lema 3.1.

Teniendo en cuenta que cada nodo conoce sus triángulos, conocerá también entonces los cliques en los que se encuentra, esto sumado a que saben el tamaño del clique máximo, les es suficiente para saber si están en él o no (o en uno de ellos).

Cuando llega un nuevo nodo, digamos w, el clique máximo ya está marcado y el tamaño es conocido, basta verificar si w forma uno más grande y actualizar la información, lo que también se puede hacer en $\mathcal{O}(n)$, ya que transmitir el tamaño al grafo se puede hacer en

$diam(G)$ rondas y w puede decirle al clique máximo quién es y avisar que los demás no lo son. □

Veamos ahora la otra cota:

Lema 3.4 *Cualquier protocolo que resuelve* MAX CLIQUE *en el* MID*, para un grafo G cualquiera, requiere $\Omega(n)$ rondas.*

Demostración. Sea G un grafo, este tendrá dos C_3 unidos entre ellos por un camino de largo $\mathcal{O}(n)$, ver Figura 3.10. Claramente el clique de tamaño máximo es K_3. Llamemos al primer ciclo C_1 y al segundo C_2.

Supongamos por contradicción que existe $\mathcal{A}$ un algoritmo que resuelve MAX CLIQUE en el modelo MID en tiempo sublineal. Sea j la etapa en la que ya es revelada la construcción mencionada anteriormente. Como $\mathcal{A}$ resuelve MAX CLIQUE en cada etapa, significa que en la etapa j, s.p.g., $\{v \in V : I(v) = 1\} = C_1$.

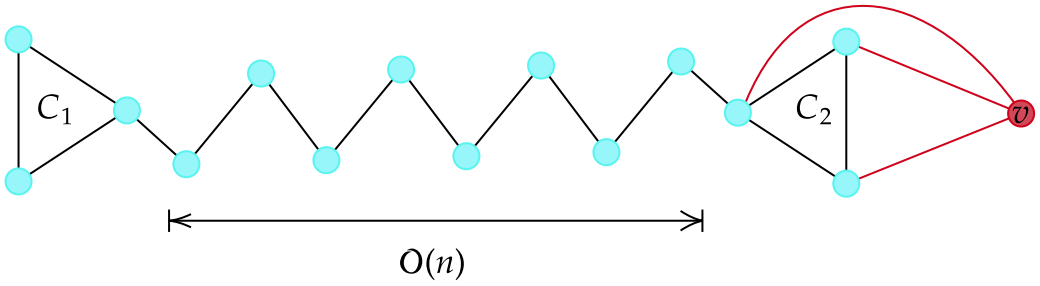

Figura 3.10: Construcción del contraejemplo para MAX CLIQUE. Dos cliques de tamaño 3, C_1 y C_2 separados por $\mathcal{O}(n)$ nodos. A uno de estos ciclos se le ancla un vértice v formando un clique de tamaño 4.

En la etapa $j + 1$ es revelado el nodo v, el cuál se conecta de manera completa a C_2. Nuevamente, como $\mathcal{A}$ resuelve MAX CLIQUE se tendrá que $\{v \in V : I(v) = 1\} = C_2 \cup v$. En particular se tendrá que $\forall v \in C_2,\ I(v) = 0$. Los únicos nodos que tienen información distinta al inicio de la etapa $j + 1$ son los nodos de C_2, pero los nodos de C_1 al terminar el algoritmo toman una decisión diferente a la etapa anterior. Luego a los nodos de C_1 se le tiene que transmitir algún mensaje mientras se ejecuta $\mathcal{A}$. Como $\mathcal{A}$ corre en tiempo sublineal y la información solo se pasa a través de las aristas, esta no puede estar a una distancia mayor a una distancia sublineal de C_2. Por lo tanto C_1 y C_2 están a una distancia sublineal. Esto es una contradicción ya que la construcción garantiza que estos están a una distancia lienal. Luego no hay algoritmo que tome menos de $\mathcal{O}(n)$ rondas en resolver MAX CLIQUE.

□

Corolario 3.1 *El problema* MAX CLIQUE *en el* MID *toma $\Theta(n)$ rondas.*

3.2.4. Comparación del MID con otros modelos

Ya fue introducido al inicio de este trabajo el modelo **BCAST**, el cual se puede pensar como si los nodos pudieran escribir mensajes en una pizarra que todos ven. Como se ha podido ver, siempre hay ciertas consideraciones importantes que pueden hacer variar y enriquecer al modelo. En este caso trabajaremos con modelos que surgen de variaciones en dos parámetros, estos son modelos definidos en [35].

El primer parámetro corresponde a la dualidad $sincrónico - asincrónico$. Los protocolos sincrónicos corresponden a aquellos en los que se descansa en aparatos externos que cercioran que la entrega de mensaje sea uno a uno, mientras que los asincrónicos tienen que lidiar con mensajes coincidentes, o sea que los mensajes son creados apenas los nodos se activan. El segundo parámetro diferenciará entre protocolos libres y simultáneos. En los simultáneos, los nodos deben estar listos para hablar en cualquier momento, mientras que en los libres pueden decidir cuando activarse.

Las cuatro combinaciones posibles de estas características darán origen a cuatro modelos diferentes que se presentan a continuación, definidos a fondo en [35]. En cada modelo, un adversario será capaz de escoger dentro de los nodos activos, cuál debe escribir en la pizarra.

- `SimAsync`: Todos los nodos se inicializan activos y crean sus mensajes al activarse.
- `FreeAsync`: Ningún nodo comienza activo y crean sus mensajes al activarse.
- `SimSync`: Todos los nodos se inicializan activos y crean sus mensajes al ser escogidos.
- `FreeSync`: Ningún nodo comienza activo y crean sus mensajes al ser escogidos.

Visto de otra manera, en el modelo `SimAsync`, los nodos deben tener su mensaje listo desde el inicio. En el modelo `FreeAsync` los nodos crean el mensaje al activarse, por lo que no es modificable incluso si pasan turnos en que no los escogen. En el modelo `SimSync` el adversario puede escoger a cualquier nodo desde el inicio pero estos pueden esperar hasta ese momento para crear su mensaje. Finalmente en el modelo `FreeSync` los nodos deciden tanto cuando activarse como cuando crear su mensaje.

Hay ciertos problemas que se usaron en [35] para comparar el poder de los diferentes modelos, dando como resultado que:

$$\texttt{SimAsync}[f(n)] \subsetneq \texttt{SimSync}[f(n)] \subsetneq \texttt{FreeAsync}[f(n)] \subseteq \texttt{FreeSync}[f(n)]$$

Donde $f(n)$ corresponde al tamaño máximo de los mensajes.

3.2.5. MIS(x)

El problema que marca la diferencia de poder entre los modelos `SimAsync` y `SimSync` es `MIS(x)` definido a continuación (para ver detalles revisar [35]).

- $\texttt{MIS(x)} = \Big\{(G, I) | I : V(G) \to \{0, 1\},\ \{v \in V(G) : I(v) = 1\}$ forman un conjunto independiente maximal y $I(x) = 1\Big\}$

Lema 3.5 *Existe un protocolo probabilista que resuelve el problema* MIS(x) *en el* MID *en* $\min\{\mathcal{O}(\log\Delta\cdot\log\log n)+2^{\mathcal{O}(\sqrt{\log\log n\cdot\log\log\log n})}, \log\Delta\cdot 2^{\mathcal{O}(\sqrt{\log\log n})}\}$ *rondas con alta probabilidad.*

Demostración. En el MID el problema depende fuertemente de si ya se encontró o no al vértice x. Si x no ha aparecido en las etapas ya pasadas, no se puede hacer nada, puesto que cualquier independiente que se escoja, puede ser completamente cambiado al aparecer x. Al momento en que aparezca el vértice x se puede resolver como en el modelo Congest, donde el mejor protocolo probabilista, hasta ahora, corre en $\min\{\mathcal{O}(\log\Delta\cdot\log\log n) + 2^{\mathcal{O}(\sqrt{\log\log n\cdot\log\log\log n})}, \log\Delta\cdot 2^{\mathcal{O}(\sqrt{\log\log n})}\}$ rondas [36]. Una vez resuelto esto, cada nodo conoce si está o no en el independiente, para cada nodo que llega posterior es trivial, si no ve a nadie del independiente se agrega, si ve alguno, no hace nada. □

Otro modelo similar al MID es el modelo OLOCAL [37]. En este los nodos son revelados de manera secuencial también, pero son capaces de revisar una vecindad de radio T y con esa información (y la de los nodos que ya han sido procesados) toman su decisión. Antes definiremos también el modelo LOCAL [3, 38], del cual se origina el modelo OLOCAL.

Definición 3.1 *[3, 38] En el modelo LOCAL de computación distribuida, el adversario etiqueta los nodos con identificadores únicos en* $\{1, 2, ..., poly(n)\}$*. Un algoritmo en el modelo LOCAL es un algoritmo distribuido en el cual cada nodo escoge en paralelo su output local basado en su vecindad de radio* T *(el output puede depender en la estructura del grafo, input de etiquetas y los identificadores únicos).*

Definición 3.2 *[37] En el modelo online LOCAL (OLOCAL) los nodos son procesados secuencialmente con respecto a a secuencia* $\sigma = v_1, v_2, ..., v_n$ *entregada por un adversario. Sea* $\sigma_1 = v_1, v_2, ..., v_i$ *que denota los primeros i nodos de la secuencia y sea*

$$G_i = G\left[\bigcup_{j=1}^{i} B(v_j, T)\right],$$

donde $B(v, T)$ *es la vecindad de radio* T *del nodo* v*, el subgrafo inducido por las vecindades de radio* T *de los primeros i nodos. Cuando el adversario presenta un nodo* v_i*, el algoritmo etiqueta a* v_i *a partir de* σ_i *y* G_i*.*

Podemos ver entonces que la principal diferencia entre MID y el modelo OLOCAL es que al ser revelado un nodo u, en MID este nodo sólo conocerá a sus vecinos. En el modelo OLOCAL cuando un nodo u es revelado este conocerá no solo su vecindad de radio T, si no que también el grafo ya revelado y la vecindad del grafo de radio T. Esto viene a partir de la otra gran diferencia entre los modelos, en el modelo OLOCAL no hay limite sobre el tamaño de los mensajes que pueden ser enviados entre nodos. Por lo tanto si al nodo solo se le diera a conocer su vecindad de radio T es irrelevante pues en una ronda se le podría comunicar

toda la otra información. Por último, en el modelo OLOCAL no hay problemas con revelar al grafo de manera tal que en cierta etapa se tenga un grafo no conexo.

Se proseguirá por estudiar dos problemas que se pueden resolver en el modelo OLOCAL y veremos como se comportan en MID. Se definirán ambos problemas y se darán protocolos para resolverlo, además de un análisis de complejidad de estas soluciones. Es importante notar que en el modelo OLOCAL y LOCAL la complejidad se mide según el tamaño del radio que se necesite.

3.2.6. Block Labeling

Definición 3.3 *(BLOCK LABELING [37]) Para el problema de BLOCK LABELING se tendrá dos conjuntos de etiquetas para los nodos, el de entrada y el de salida. El conjunto de etiquetas de entrada es* $\Sigma = \{a, b, c\}$ *y el conjunto de etiquetas de salida es* $\Gamma = (1, 2, 3, x)$*. Un etiquetado* L *es una solución factible para el input* (G, I) *si para cada arista* $uv \in E$ *se cumple que:*

BL1 Si $I(u) = a$ *y* $I(v) = b$ *entonces* $\{L(u), L(v)\} = \{1, x\}$

BL2 Si $I(u) = a$ *y* $I(v) = c$ *entonces* $\{L(v), L(u)\} = \{2, x\}$

BL3 Si $I(u) = b$ *y* $I(v) = c$ *entonces* $\{L(u), L(v)\} = \{3, x\}$

BL4 Si $I(u) = I(v)$ *entonces* $L(u) = L(v)$

Vease la Figura 3.11 para un ejemplo de un par input-output.

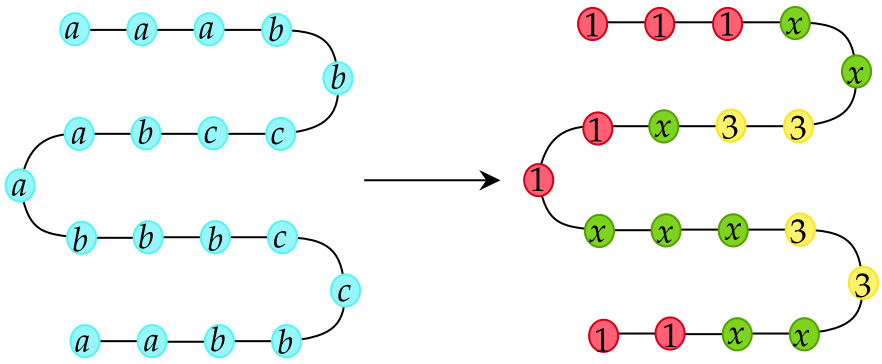

Figura 3.11: El problema BLOCK LABELING. A la izquierda: un input I. A la derecha: un posible output L

Definición 3.4 *La promesa de dos colores corresponde a cuando el etiquetado de entrada* $I : V \to \Sigma$ *es tal que* $|I(V)| \leq 2$*, esto es al menos un color no aparece en la entrada.*

Veamos como resuelve este problema el modelo OLOCAL. Se dará el protocolo correspondiente pero para más detalles revisar [37].

Lema 3.6 *El problema BLOCK LABELING con promesa de dos colores en caminos se puede resolver en el modelo OLOCAL con un radio 0.*

Demostración. Veamos el protocolo:

Protocolo BLOCK LABELING en modelo OLOCAL

1. Se etiqueta al primer nodo v_1 con la etiqueta $L(v_1) = x$.
2. Cuando por primera vez se ve a un nodo v_i tal que $I(v_1) \neq I(v_1)$, se escoge una etiqueta $L(v_i)$ que cumpla las restricciones (BL1)-(BL3) incluso si v_1 y v_i son adyacentes.
3. Cada nodo con etiqueta de color $I(v_1)$ serán etiquetados con $L(v_1)$ mientras que los con color $I(v_i)$ serán etiquetados con $L(v_i)$.

Mientras hayan 2 colores, el etiquetado es claramente factible. □

Veamos como enfrentar este problema en MID.

Lema 3.7 *Existe un protocolo que resuelve el problema BLOCK LABELING con promesa de dos colores en caminos en MID.*

Demostración. A diferencia que en el modelo OLOCAL, donde cada nodo conoce el grafo ya revelado, en el MID Los nodos sólo conocen a sus vecinos al ser revelados. En ese sentido, si un nodo aislado fuese revelado, no sería posible etiquetarlo sin correr el riesgo de que al conectarlo con el resto del grafo este etiquetado no sea valido. Pero por como es definido este modelo, esto no es posible. Los nodos son siempre revelados de manera que el grafo sea conexo.

Es posible entonces explotar la estructura del camino para generar un protocolo:

Protocolo BLOCK LABELING en modelo OLOCAL

1. Se etiqueta al primer nodo v_1 con la etiqueta $L(v_1) = x$.
2. Cuando por primera vez se ve a un nodo v_i tal que $I(v_1) \neq I(v_1)$, se escoge una etiqueta $L(v_i)$ que cumpla las restricciones (BL1)-(BL3).
3. Cada nodo con etiqueta de color $I(v_1)$ serán etiquetados con $L(v_1)$ mientras que los con color $I(v_i)$ serán etiquetados con $L(v_i)$.

Replicar el protocolo dado en OLOCAL es posible por la conexidad del grafo en todo momento. Si en MID se diera la situación de no conexidad, entonces no sería posible resolver el problema en menos de $\Omega(n)$ pues en cada iteración se tendría que corregir todo. Esto es porque podría dar el caso en que si $P = v_1, ..., v_n$ es el camino que se presenta, los nodos sean revelados alternadamente, v_1, v_n, v_2, v_{n-1} y así sucesivamente. Luego no se puede asignar color a v_1 y a v_n hasta que el camino este conectado y se comuniquen sus respectivas etiquetas. □

Es interesante notar como puede hacer tanta diferencia un pequeño cambio en las condiciones impuestas en el modelo.

3.2.7. Vertex Coloring

Uno de los problemas más relevantes de la computación distribuida y la teoría de grafos en general, es el del la coloración de vértices de un grafo VERTEX-COL. Si el problema se centra en ciertos grafos en específico, es posible ver cuál es la cantidad mínima de colores que se necesita para poder colorear a este, o cuántos colores necesita el modelo para poder hacerlo.

En el caso de las grillas 2-dimensionales, es fácil notar que se pueden colorear con 2 colores, sin embargo bajo ciertos modelos este problema no es fácil de resolver. Tanto para el modelo LOCAL como el OLOCAL es directo que se pueden colorear grillas con 5 y 4 colores, pero es el caso de 3 colores el que los diferencia. Mientras en el modelo LOCAL es necesario conocer una vecindad de tamaño $\mathcal{O}(\sqrt{n})$ (que es el diámetro de una grilla de n nodos), en el modelo OLOCAL solo es necesario un radio de $\mathcal{O}(\log n)$ [37]. Por otro lado, en MID es un problema que se puede resolver de forma directa.

Lema 3.8 *El número de colores que se necesita para colorear una grilla 2-dimensional G de n nodos en el Modelo Dinámico es 2 y toma $\mathcal{O}(1)$ rondas, es decir un número constante de rondas.*

Demostración. Manteniéndose fiel al modelo, se tendrá la promesa de que se está trabajando en una grilla y que los nodos aparecerán de manera tal que el grafo se mantenga conexo. Para esto existe un protocolo para pintar con solo 2 colores.

Protocolo coloreamiento grilla dos dimensional
1. El primer nodo en aparecer, v_1, escoge el color 0.
2. Cada nodo en adelante escoge el color opuesto al de sus vecinos.

Al ser este un grafo bipartito, cualquier subgrafo también lo será. Luego, las elecciones de los colores siempre serán validas y correctas.

Similar al lema anterior, en el caso en el que no se pueda contar con un grafo conexo se topa con las mismas barreras que antes, la falta de coordinación es crítica y la complejidad escala a $\mathcal{O}(\sqrt{n})$ rondas, de manera similar al modelo LOCAL. Lamentablemente es necesario esperar a que los nodos se conecten para calzar en los colores que se escogen. □

Conclusión

En el presente trabajo se estudiaron dos temas que involucran teoría de grafos y computación distribuida. En primer lugar, se estudió el problema de calcular el diámetro para diferentes clases de grafos en distintos modelos. A continuación se vio cómo estos resultados podían generalizarse para otras clases de grafos y problemas. Finalmente se presentó y examinó un nuevo modelo de computación distribuida que se denominó MID.

La primera parte de esta tesis consistió en la caracterización del problema del cálculo del diámetro para grafos de intervalos, reduciendo el problema a calcular la distancia entre los nodos cuyos intervalos están más lejos en alguna representación de intervalos del grafo. Luego de esto se encontró un método para la reconstrucción de la representación de intervalos del grafo y a partir de esto se presentó un PLS para el problema, el cual usa certificados de tamaño $\mathcal{O}(\log n)$. Posterior a esto, en el modelo `Congest` se presentó un protocolo probabilista que permite a los nodos de un grafo split decidir si pertenecen al clique o al independiente. Con esto se desarrolló un algoritmo para el cálculo del diámetro cuando el clique es a lo más $\mathcal{O}(\log \log n)$ o cuando el independiente es a lo más $\mathcal{O}(\sqrt{n})$. En el caso de un grafo split balanceado se demostró que con alta probabilidad se resuelve el problema del cálculo del diámetro en $\mathcal{O}(n^{0.158} \log n)$ rondas también el modelo `Congest`. A continuación, se resolvió el problema de la reconstrucción de grafos en familias de grafos hereditarias similares a la de los split en el modelo `Congest` con un protocolo de $\mathcal{O}(\frac{\log |\mathcal{G}_n|}{n \log n} + f(n))$ rondas si el problema se puede resolver en $f(n)$ rondas en el modelo `Congested Clique`.

En la segunda parte de esta tesis se introdujo un nuevo modelo de computación distribuida llamado MID. Se estudió el comportamiento de este modelo en problemas que tenían soluciones locales como la elección del líder o el matching máximo, donde se describieron algoritmos que corren en un número constante de rondas. Luego se expandió este trabajo a la detección de subgrafos de 4 nodos donde se resolvió para cualquier estructura de 4 nodos conexa en $\mathcal{O}(n)$ rondas. Posterior a esto se demostró que el problema de la identificación del clique máximo no podía ser resuelto en menos de n rondas demostrando que el modelo no es todopoderoso y por lo tanto si es interesante de estudiar. Por último se presentaron problemas de modelos similares, como MIS(x) en modelos del tipo con pizarra o la coloración de grillas 2-dimensionales, en el modelo OLOCAL y se revisó cómo se comparaba nuestro modelo con los otros en estos problemas.

Esta tesis logra resolver parcialmente el problema del cálculo del diámetro en el modelo `Congest` para los grafos split. Por un lado resuelve algunos casos, dentro de los cuales se encuentra el caso balanceado. El caso general es más desafiante porque el clique representa la menor parte del grafo. Esto conlleva a que gran parte del grafo sean nodos aislados entre ellos con una mayor saturación de los canales de comunicación. Como trabajo futuro se podría

indagar en el caso general e incluso explorar una familia más general: los grafos cordales.

Además, dentro de la generalización de los grafos split el algoritmo encontrado se enfoca en familias hereditarias, por lo que una búsqueda para familias cualquiera es muy interesante. También mejorar el resultado encontrado, pues este depende de conocer la familia y tener una identificación de cuál es el clique del grafo.

Una pregunta que queda abierta es la importancia del MID en el campo de la computación distribuida, ¿cómo se comporta con problemas clásicos y qué tan relevante es para aplicaciones presentes o futuras en el mundo real? Una importante pregunta a resolver es cómo se relaciona este modelo con otros. Si bien tiene resultados comparables con modelos como OLOCAL, características como la conexidad en los grafos juegan un rol fundamental, pues esencialmente define modelos diferentes. Luego, es importante plantear preguntas como por ejemplo ¿Es MID más poderoso que el modelo OLOCAL? o viceversa.

Bibliografía

[1] Pal, M., "Intersection graphs: An introduction," CoRR, vol. abs/1404.5468, 2014, http://arxiv.org/abs/1404.5468.

[2] Ducoffe, G., Habib, M., y Viennot, L., "Fast diameter computation within split graphs," CoRR, vol. abs/1910.03438, 2019, http://arxiv.org/abs/1910.03438.

[3] Peleg, D., Distributed Computing: A Locality-Sensitive Approach. USA: Society for Industrial and Applied Mathematics, 2000.

[4] Lotker, Z., Patt-Shamir, B., Pavlov, E., y Peleg, D., "Minimum-weight spanning tree construction in o(log log n) communication rounds," SIAM Journal on Computing, vol. 35, no. 1, pp. 120–131, 2005, doi:10.1137/S0097539704441848.

[5] Drucker, A., Kuhn, F., y Oshman, R., "The communication complexity of distributed task allocation," en Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing, PODC '12, (New York, NY, USA), p. 67–76, Association for Computing Machinery, 2012, doi:10.1145/2332432.2332443.

[6] Drucker, A., Kuhn, F., y Oshman, R., "On the power of the congested clique model," en Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, PODC '14, (New York, NY, USA), p. 367–376, Association for Computing Machinery, 2014, doi:10.1145/2611462.2611493.

[7] Hegeman, J. W., Pandurangan, G., Pemmaraju, S. V., Sardeshmukh, V. B., y Scquizzato, M., "Toward optimal bounds in the congested clique: Graph connectivity and mst," en Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC '15, (New York, NY, USA), p. 91–100, Association for Computing Machinery, 2015, doi:10.1145/2767386.2767434.

[8] Kor, L., Korman, A., y Peleg, D., "Tight bounds for distributed minimum-weight spanning tree verification," Theory of Computing Systems, vol. 53, pp. 318–340, 2013.

[9] Ghaffari, M. y Parter, M., "Mst in log-star rounds of congested clique," en Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC '16, (New York, NY, USA), p. 19–28, Association for Computing Machinery, 2016, doi:10.1145/2933057.2933103.

[10] Montealegre, P., Perez-Salazar, S., Rapaport, I., y Todinca, I., "Graph reconstruction in the congested clique," CoRR, vol. abs/1706.03107, 2017, http://arxiv.org/abs/1706.03107.

[11] Halldórsson, M. M., Kuhn, F., y Maus, Y., "Distance-2 coloring in the CONGEST model," CoRR, vol. abs/2005.06528, 2020, https://arxiv.org/abs/2005.06528.

[12] Frischknecht, S., Holzer, S., y Wattenhofer, R., "Networks cannot compute their diameter

in sublinear time," en Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12, (USA), p. 1150–1162, Society for Industrial and Applied Mathematics, 2012.

[13] Holzer, S. y Wattenhofer, R., "Optimal distributed all pairs shortest paths and applications," en Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing, PODC '12, (New York, NY, USA), p. 355–364, Association for Computing Machinery, 2012, doi:10.1145/2332432.2332504.

[14] Grossman, O., Khoury, S., y Paz, A., "Improved Hardness of Approximation of Diameter in the CONGEST Model," en 34th International Symposium on Distributed Computing (DISC 2020) (Attiya, H., ed.), vol. 179 de Leibniz International Proceedings in Informatics (LIPIcs), (Dagstuhl, Germany), pp. 19:1–19:16, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, doi:10.4230/LIPIcs.DISC.2020.19.

[15] Abboud, A., Censor-Hillel, K., y Khoury, S., "Near-linear lower bounds for distributed distance computations, even in sparse networks," CoRR, vol. abs/1605.05109, 2016, http://arxiv.org/abs/1605.05109.

[16] Brandstädt, A., Le, V. B., y Spinrad, J. P., Graph Classes: A Survey. Society for Industrial and Applied Mathematics, 1999, doi:10.1137/1.9780898719796.

[17] Ramírez Romero, D. N., "Detección de clases de grafos en el modelo interactivo de verificación distribuida," 2020.

[18] Hammer, P. L. y Simeone, B., "The splittance of a graph," Combinatorica, vol. 1, pp. 275–284, 1981.

[19] Becker, F., Montealegre, P., Rapaport, I., y Todinca, I., "The role of randomness in the broadcast congested clique model," Inf. Comput., vol. 281, 2021, doi:10.1016/j.ic.2020.104669.

[20] Kushilevitz, E., "Communication complexity," vol. 44 de Advances in Computers, pp. 331–360, Elsevier, 1997, doi:https://doi.org/10.1016/S0065-2458(08)60342-3.

[21] Ghaffari, M. y Nowicki, K., "Congested clique algorithms for the minimum cut problem," en Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC '18, (New York, NY, USA), p. 357–366, Association for Computing Machinery, 2018, doi:10.1145/3212734.3212750.

[22] Censor-Hillel, K., Kaski, P., Korhonen, J. H., Lenzen, C., Paz, A., y Suomela, J., "Algebraic methods in the congested clique," CoRR, vol. abs/1503.04963, 2015, http://arxiv.org/abs/1503.04963.

[23] Le Gall, F., "Powers of tensors and fast matrix multiplication," en Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC '14, (New York, NY, USA), p. 296–303, Association for Computing Machinery, 2014, doi:10.1145/2608628.2608664.

[24] Censor-Hillel, K., Kaski, P., Korhonen, J. H., Lenzen, C., Paz, A., y Suomela, J., "Algebraic methods in the congested clique," CoRR, vol. abs/1503.04963, 2015, http://arxiv.org/abs/1503.04963.

[25] Nanongkai, D., "Distributed approximation algorithms for weighted shortest paths," en Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, STOC '14, (New York, NY, USA), p. 565–573, Association for Computing Machinery,

2014, doi:10.1145/2591796.2591850.

[26] Das Sarma, A., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., y Wattenhofer, R., "Distributed verification and hardness of distributed approximation," en Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC '11, (New York, NY, USA), p. 363–372, Association for Computing Machinery, 2011, doi:10.1145/1993636.1993686.

[27] Björklund, A., Kaski, P., y Kowalik, L., "Probably Optimal Graph Motifs," en 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013) (Portier, N. y Wilke, T., eds.), vol. 20 de Leibniz International Proceedings in Informatics (LIPIcs), (Dagstuhl, Germany), pp. 20–31, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, doi:10.4230/LIPIcs.STACS.2013.20.

[28] Gall, F. L., "Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems," CoRR, vol. abs/1608.02674, 2016, http://arxiv.org/abs/1608.02674.

[29] Lenzen, C., "Optimal deterministic routing and sorting on the congested clique," CoRR, vol. abs/1207.1852, 2012, http://arxiv.org/abs/1207.1852.

[30] Czumaj, A., Davies, P., y Parter, M., "Simple, deterministic, constant-round coloring in the congested clique," CoRR, vol. abs/2009.06043, 2020, https://arxiv.org/abs/2009.06043.

[31] Saikia, P. y Karmakar, S., "Distributed approximation algorithms for steiner tree in the CONGESTED CLIQUE," CoRR, vol. abs/1907.12011, 2019, http://arxiv.org/abs/1907.12011.

[32] Afek, Y., Awerbuch, B., y Gafni, E., "Applying static network protocols to dynamic networks," en Proceedings of the 28th Annual Symposium on Foundations of Computer Science, SFCS '87, (USA), p. 358–370, IEEE Computer Society, 1987, doi:10.1109/SFCS.1987.7.

[33] Awerbuch, B., Cidon, I., y Kutten, S., "Optimal maintenance of a spanning tree," J. ACM, vol. 55, 2008, doi:10.1145/1391289.1391292.

[34] Augustine, J., Pandurangan, G., y Robinson, P., "Fast byzantine leader election in dynamic networks," en Distributed Computing (Moses, Y., ed.), (Berlin, Heidelberg), pp. 276–291, Springer Berlin Heidelberg, 2015.

[35] Becker, F., Kosowski, A., Nisse, N., Rapaport, I., y Suchan, K., "Allowing each node to communicate only once in a distributed system: Shared whiteboard models," Distributed Computing, vol. 28, 2012, doi:10.1145/2312005.2312008.

[36] Ghaffari, M., "Distributed maximal independent set using small messages," en Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '19, (USA), p. 805–820, Society for Industrial and Applied Mathematics, 2019.

[37] Akbari, A., Lievonen, H., Melnyk, D., Sarkijarvi, J., y Suomela, J., "Online algorithms with lookaround," CoRR, vol. abs/2109.06593, 2021, https://arxiv.org/abs/2109.06593.

[38] Linial, N., "Locality in distributed graph algorithms," SIAM Journal on Computing, vol. 21, no. 1, pp. 193–201, 1992, doi:10.1137/0221015.