Universidad Central “Marta Abreu” de Las Villas
Facultad de Matemática, Física y Computación

Trabajo para optar por el Título Académico de

Máster en Ciencia de la Computación

Algoritmo dinámico para el cálculo de la intermediación

Autor:
Lic. Reynaldo Gil Pons

Tutora:
Dra. Leticia Arco García

2019

Ubi Dubium ibi Libertas
PROVERBIO LATINO

Hago constar que el presente Trabajo para optar por el Título de Máster en Ciencia de la Computación ha sido realizado en la facultad de Matemática, Física y Computación de la Universidad Central "Marta Abreu" de Las Villas (UCLV) como parte de la culminación de los estudios de la Maestría en Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución para los fines que estime conveniente, tanto de forma total como parcial y que además no podrá ser presentado en eventos ni publicado sin la previa autorización de la UCLV.

_____

Firma del Autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____

Firma del Tutor

Resumen

La intermediación es una de las medidas de centralidad más populares en el análisis de redes sociales. En la actualidad, estas redes son usualmente grandes y dinámicas. Calcular la intermediación en tales redes ha demostrado ser una tarea algorítmica muy compleja. En este trabajo, proponemos un nuevo algoritmo dinámico para el cálculo de la intermediación de todos los nodos en grafos dirigidos extraídos de redes sociales. Nuestro algoritmo usa un espacio en memoria lineal, lo que lo hace adecuado para aplicaciones de gran escala. La evaluación experimental en una variedad de redes reales y sintéticas ha mostrado que nuestro algoritmo es más rápido que realizar todos los cálculos desde cero y que además es competitivo con propuestas recientes.

Palabras clave: Análisis de Redes Sociales, Intermediación, Algoritmos Dinámicos, Grafos Dinámicos.

Abstract

Betweenness is one of the most popular centrality measures in the analysis of social networks. Nowadays, these networks are usually big and dynamic. Computing betweenness in such networks has become a very complex algorithmic task. In this work, we propose a new dynamic algorithm to compute betweenness centrality of all nodes in directed graphs extracted from social networks. Our algorithm uses linear space, making it suitable for large scale applications. The experimental evaluation on a variety of real-world and synthetic networks has shown our algorithm is faster than recalculation from scratch and competitive with recent approaches.

Keywords: Social Network Analysis, Betweenness Centrality, Dynamic Algorithms, Dynamic Graphs.

Índice general

Introducción

Las redes sociales son estructuras sociales compuestas por un conjunto de actores (tales como individuos y organizaciones) y un conjunto de relaciones entre estos actores [10]. En la actualidad, gran parte de las comunicaciones humanas se producen a partir de medios digitales, interacciones que pueden ser estudiadas de manera natural como redes sociales. Por ello, la extracción de información y conocimientos a partir de las redes sociales se ha convertido en una tarea de gran impacto social y económico.

En años recientes, las redes sociales han cobrado gran popularidad, debido a la aparición de Internet hace más de dos décadas, permitiendo la creación de lo que hoy llamamos sitios de redes sociales; y a la proliferación y disponibilidad de dispositivos conectados a Internet tales como las computadoras personales y los teléfonos celulares [10]. Ya en el año 2000, con el surgimiento de la WEB 2.0 y todas sus funcionalidades subyacentes (sobre todo la creación de contenido por parte de los usuarios), las redes sociales en Internet cobraron gran fuerza. Hoy en día la interacción con redes sociales ha ganado un impresionante crecimiento. Solo por mencionar algunos ejemplos, actualmente Facebook tiene 1280 millones de usuarios activos diariamente y en total más de 1940 millones de usuarios activos mensualmente; así mismo, Twitter tiene 313 millones de usuarios activos por mes y 500 millones de *tweets* diarios[1]. Esto ha propiciado que en la actualidad las redes sociales sean utilizadas como ba-

[1]http://www.socialbakers.com/statistics/ Visitado el 25 de abril de 2019

se para desarrollar diversas aplicaciones de marketing, dándoles así una relevancia apreciable desde el punto de vista comercial. Además, son un método muy eficiente de difusión de información, por lo que son utilizadas por la mayoría de las agencias noticiosas para este propósito. Todo lo antes expuesto ha hecho necesario diseñar algoritmos y métodos que nos permitan extraer información relevante de las redes sociales existentes.

El estudio de las redes sociales comenzó a principios del siglo XX. Las primeras investigaciones fueron realizadas mayormente por especialistas de áreas como la Psicología, la Psiquiatría y la Antropología. Al menos tres grupos de investigadores comenzaron a trabajar en este tema en la década de los 30, aunque ninguno de ellos produjo resultados reconocidos globalmente [39]. No fue hasta 1970 cuando el profesor Harrison C. White de la Universidad de Harvard y sus estudiantes publicaron teorías e investigaciones de relevancia en redes sociales. A partir de entonces, los científicos de todo el mundo, sin importar el campo de investigación, no pudieron continuar ignorando estas ideas [22]. En los últimos 50 años, los métodos de redes sociales se desarrollaron como una parte integral de los avances en la teoría social, la investigación empírica, los métodos formales matemáticos y las estadísticas [56].

El Análisis de Redes Sociales (*Social Network Analysis*; SNA) es una disciplina que emplea el uso de la teoría de redes para analizar las redes sociales [1]. El SNA es aplicable en disímiles escenarios, estando una gran parte de las redes relacionadas con la WWW, entre las que encontramos los blogs y los enlaces entre ellos [9], las redes de correos electrónicos [1, 52] y las redes sociales en línea [38, 35]. Sin embargo, sus aplicaciones no están limitadas a redes de personas, sino que se pueden extrapolar a redes tales como las que se forman en las comunicaciones y transporte [54, 46, 18], y las que aparecen en las ciencias biológicas [59, 32, 40].

En la disciplina SNA las redes sociales usualmente se modelan por medio de grafos cuyos nodos representan personas u otras entidades dentro del contexto social, y cuyas aristas representan interacción, colaboración o influencia entre sus nodos. Los nodos pueden ser de uno o varios tipos, además de presentar atributos propios de los

actores de la red, tales como nombre, lugar de procedencia, edad, entre otros. Las aristas pueden ser pesadas o no; unidireccionales o bidireccionales, dependiendo de la naturaleza de la red social en cuestión. Por ejemplo, si modeláramos la red social Facebook, podríamos representar las personas como nodos, siendo su nombre, fecha de nacimiento y páginas preferidas sus atributos; las aristas representarían la amistad, por lo que serían bidireccionales, y podrían ser pesadas de acuerdo a la cantidad de mensajes y comentarios intercambiados por el par de personas correspondiente.

El SNA incluye una gran diversidad de tareas, tales como: análisis de centralidad, detección de comunidades, análisis de la difusión de información, predicción de enlaces, entre otras [3]. En el presente trabajo nos ocuparemos específicamente del Análisis de Centralidad. En la teoría de grafos y el SNA, la centralidad de un nodo (o arista) es una medida de su importancia relativa dentro de un grafo [6]. Esta medida está directamente relacionada con la capacidad de un nodo (o arista) de hacer llegar información de un lado a otro de la red. La centralidad es uno de los conceptos más estudiados en el SNA, y fue propuesto inicialmente por Bavelas en 1940 [4]. Entre las diferentes medidas de centralidad existentes, la intermediación es una de las más populares y costosas computacionalmente. Esta medida fue introducida por Linton Freeman en 1977 [21], como forma de cuantificar qué tanto control tiene un nodo sobre la comunicación entre otros dos nodos en una red social. Los nodos que poseen alta intermediación son de cierta forma los más capaces de controlar los flujos de información que transitan en la red. Estos tienen la capacidad de bloquear e incluso de influir en las informaciones que se transmiten entre los nodos que contribuyeron a su alto valor.

En los últimos años las redes sociales han tenido un gran crecimiento en cuanto a tamaño y dinamismo [51, 60, 34]. Los algoritmos estáticos en general no tienen un buen desempeño en estas redes, pues estos procesan la red en su totalidad incluso cuando ocurren pocos cambios en esta. Debido a esta razón, los investigadores han propuesto algoritmos dinámicos para el cálculo de medidas costosas como la intermediación ya que se adaptan mejor a las condiciones de variabilidad de las redes

[55, 49, 5, 31, 36]. Estos algoritmos utilizan los resultados previos para mejorar la eficiencia computacional, permitiendo así que redes sociales mucho mayores pasen a ser tratables. En general, los algoritmos dinámicos propuestos tienen una complejidad espacial cuadrática, lo que dificulta su aplicación práctica. Recientemente fue propuesto en [31] un algoritmo dinámico con complejidad espacial lineal para el cálculo de la intermediación. Este solo puede ser utilizado con grafos no dirigidos por lo que no es posible su aplicación en redes sociales donde las relaciones tienen dirección.

El **problema científico** que nos ocupa en la presente tesis es la ausencia de algoritmos eficientes en tiempo y espacio para calcular la intermediación de cada uno de los nodos de un grafo dirigido dinámico.

Para contribuir a la solución del problema científico antes planteado, se formuló la siguiente **hipótesis** de investigación: La incorporación de la descomposición en componentes biconexos en el cálculo de la intermediación permitirá obtener un algoritmo con menor complejidad espacial y temporal, y por tanto, su aplicación en grafos dinámicos dirigidos provenientes de las redes sociales.

En conformidad con la hipótesis de investigación identificada, el **objetivo general** de la investigación consiste en desarrollar un algoritmo dinámico eficiente con complejidad espacial lineal capaz de calcular la intermediación en grafos dinámicos y dirigidos extraídos de redes sociales.

Este objetivo general fue desglosado en los **objetivos específicos** siguientes:

- Identificar las características, ventajas y desventajas principales de los algoritmos dinámicos existentes para calcular la intermediación en redes sociales.
- Diseñar e implementar un algoritmo para calcular la intermediación en grafos dinámicos, buscando la menor complejidad temporal y espacial posible.
- Evaluar la eficiencia del algoritmo mediante experimentos en grafos sintéticos

y reales.

La **novedad científica** principal que aporta esta investigación radica en el diseño de un algoritmo dinámico para el cálculo de la intermediación con una complejidad espacial lineal, permitiendo procesar grafos de gran tamaño y dirigidos de una manera eficiente desde el punto de vista temporal y espacial.

El **valor teórico** de la investigación está directamente relacionado con su novedad científica, al hallarse una forma más eficiente del cálculo de la intermediación.

El **valor práctico** de la tesis radica en dotar a los analistas de un nuevo método computacional que permite procesar datos relevantes de una red social de una manera menos costosa.

La presente tesis está compuesta por introducción, tres capítulos y conclusiones. En el Capítulo 1 se describen los fundamentos teóricos necesarios para la correcta comprensión de la tesis, tales como conceptos de redes sociales y las formas de calcular la intermediación. En el Capítulo 2 se presenta el algoritmo propuesto y se analiza la eficiencia del mismo. En el Capítulo 3 se muestran los experimentos realizados y se discuten los resultados obtenidos. La tesis culmina con las conclusiones, seguidamente las recomendaciones derivadas de la investigación y las referencias bibliográficas.

Capítulo 1

Acerca del cálculo de la intermediación en redes sociales

En este capítulo se describen las principales características de las redes sociales. Luego se realiza un resumen de los enfoques utilizados para el cálculo de la intermediación, haciendo énfasis en los más eficientes.

1.1. Fundamentos

A continuación presentamos las definiciones de grafos [12] que serán necesarias para la correcta comprensión de nuestro trabajo.

Definición 1. Un grafo G es una estructura compuesta por un conjunto V de elementos llamados nodos, y un conjunto E de relaciones (o pares de nodos) llamadas aristas.

Definición 2. Un grafo se denomina dirigido cuando las relaciones entre los nodos son unidireccionales, y se denomina no dirigido cuando las relaciones son bidireccio-

nales.

Definición 3. Se denomina grado de un nodo a la cantidad de aristas adyacentes a este.

Definición 4. Un camino es una lista de nodos tal que cada nodo está conectado mediante una arista al siguiente nodo de la lista. La longitud del camino es la cantidad de aristas que este contiene.

Definición 5. Un ciclo es un camino que no tiene nodos repetidos a excepción del primer nodo que coincide con el último.

Definición 6. Un camino mínimo del nodo a al nodo b es cualquier camino cuya lista de nodos comienza con a y termina con b, y su longitud es mínima.

Definición 7. Un grafo no dirigido es conexo cuando entre cualquier par de nodos existe un camino.

Definición 8. Un subgrafo es un grafo formado por un subconjunto de nodos y aristas de un grafo.

Definición 9. Un subgrafo inducido es un subgrafo conformado por un conjunto de nodos y todas las aristas del grafo que conectan los nodos en el conjunto.

Definición 10. Un árbol es un grafo conexo no dirigido que no contiene ciclos, o un grafo dirigido donde existe un único camino entre un nodo (raíz) y el resto de los nodos (este caso se denomina árbol enraizado).

Definición 11. Un bosque es un grafo no dirigido que no contiene ciclos, o un grafo dirigido que está compuesto por la unión disjunta de árboles enraizados.

Definición 12. Un grafo dirigido acíclico (*Directed Acyclic Graph*; DAG) es un grafo con aristas dirigidas que no contiene ciclos.

Definición 13. En un grafo conexo un árbol de caminos mínimos respecto a un nodo origen s es un subgrafo que contiene exactamente un camino mínimo de s a cada uno de los restantes nodos del grafo.

Definición 14. El DAG de caminos mínimos de un grafo respecto a un nodo origen s es el subgrafo que contiene todas las aristas que pertenecen a algún camino mínimo de s a algún otro nodo.

Para alcanzar una mejor comprensión y simplicidad utilizaremos las siguientes notaciones matemáticas:

- $n = |V|$ es la cantidad de nodos y $m = |E|$ la cantidad de aristas de un grafo $G = (V, E)$.
- $dist_{ab}$ es la longitud de un camino mínimo del nodo a al nodo b.
- σ_{ab} es la cantidad de caminos mínimos del nodo a al nodo b.
- $\sigma_{ab}(c)$ es la cantidad de caminos mínimos del nodo a al nodo b que pasan por el nodo c.

1.1.1. Componentes biconexos

Los componentes biconexos fueron propuestos por primera vez como una buena heurística para acelerar el cálculo de la intermediación en [47], y más recientemente en el contexto de los grafos dinámicos en [31]. A continuación ofrecemos algunas definiciones y resultados relevantes relacionados con los componentes biconexos:

Definición 15. Sea G un grafo no dirigido. Un componente biconexo es un subgrafo inducido conexo A de G, tal que la eliminación de cualquier nodo no desconecta A, y es maximal.

Definición 16. Cualquier nodo que pertenece a más de un componente biconexo se denomina punto de articulación.

Lema 1. Cada arista de un grafo pertenece exactamente a un componente biconexo, por lo que dos componentes biconexos cualesquiera solo pueden tener nodos en común.

Demostración. Supongamos lo contrario, que una arista (u, v) pertenece a dos componentes biconexos distintos B_1 y B_2 del grafo. Como los componentes biconexos son subgrafos inducidos, entonces u y v pertenecen a ambos componentes también. Analicemos el subgrafo inducido B que se forma partiendo de la unión de los nodos de los dos componentes biconexos. Probaremos que en este subgrafo, la eliminación de un nodo cualquiera no desconecta el grafo, lo cual sería una contradicción ya que los componentes biconexos iniciales debían ser maximales. Supongamos que se elimina un nodo x. Sin pérdida de generalidad, podemos asumir que $x \neq u$. Luego, u está conectado al resto de los nodos de B, ya que B_1 y B_2 son componentes biconexos, y u pertenece a ambos. Por tanto, eliminar x de B no desconecta este componente, como se quería demostrar. □

Lema 2. Si definimos un nuevo grafo donde los nodos representan componentes biconexos, y dos nodos están conectados si y solo si los correspondientes componentes biconexos no son disjuntos, entonces este nuevo grafo es un bosque.

Demostración. Para demostrar que este nuevo grafo es un bosque, basta demostrar que no tiene ciclos. Supongamos lo contrario. Por tanto, el subgrafo inducido por la unión de todos los componentes biconexos en el ciclo, cumple la propiedad de que cuando se elimina cualquiera de sus nodos, el grafo se mantiene conexo. Luego, los componentes biconexos en el ciclo no eran maximales, lo cual es una contradicción. □

Corolario 1. De acuerdo al lema anterior, todos los caminos simples entre dos nodos (si existe al menos uno) en diferentes componentes biconexos comparten al menos un punto de articulación en el componente del nodo inicial del camino, y uno en el componente del nodo final.

1.1.2. Grafos dinámicos

Muchas redes sociales son inherentemente dinámicas [37, 24]; con el paso del tiempo aparecen nuevas relaciones entre los actores, o son eliminadas relaciones existentes. Las redes dinámicas se representan matemáticamente a través de grafos dinámicos. Según los tipos de modificaciones, estos grafos pueden ser incrementales (se agregan aristas), decrementales (se eliminan aristas), o dinámicos (ambas posibles modificaciones).

Los grafos dinámicos han sido ampliamente estudiados en las últimas décadas [27]. Excepto algunos casos triviales, los algoritmos que procesan grafos dinámicos son usualmente mucho más complicados que sus contrapartes estáticas. Algunas técnicas generales como el agrupamiento, la transformación de los grafos en dispersos, y la utilización de algoritmos randomizados, han sido de gran valor para la obtención de algoritmos eficientes [14]. Para algunos problemas como el de identificar si un grafo no dirigido es conexo, determinar el árbol de expansión mínimo e identificar los componentes biconexos, algoritmos eficientes (en el sentido que son mucho más rápidos que sus contrapartes estáticas) han sido diseñados [29]; para otros como el cálculo de la clausura transitiva [13] y de los caminos mínimos entre todos los pares de nodos [15, 50, 45] no ha sido el caso.

Los algoritmos para el cálculo de la intermediación que procesan grafos incrementales son de cierta forma más simples que los que procesan grafos decrementales o dinámicos en general, ya que el problema de actualizar las distancias entre todos los pares de nodos es notablemente más sencillo en este caso [42]. Aún así, incluso para el caso incremental, la complejidad temporal en el peor caso de los algoritmos propuestos no es mejor que sus contrapartes estáticas. Sin embargo, mejoras sustanciales se han logrado en la gran mayoría de los casos prácticos [33, 5, 25, 31].

1.2. Redes sociales y sus principales propiedades

Una red social consiste en actores (por ejemplo, personas y organizaciones) y alguna forma de relación entre estos. La estructura de la red es usualmente modelada como un grafo, donde los vértices representan actores, y las aristas la existencia de una relación entre dos actores [8]. Estos grafos pueden tener, en general, pesos en los nodos o aristas, dirección en las aristas, aristas múltiples y lazos. Además, pueden ser dirigidos o no dirigidos. Una arista no dirigida implica que no existe distinción entre los nodos asociados a cada arista; en otras palabras, que no hay dirección en la relación que conecta a los dos nodos. Este concepto es bien ejemplificado con la relación de amistad entre personas. Por otra parte, las aristas dirigidas representan la dirección en la relación entre dos vértices distintos, como por ejemplo las relaciones de padre a hijo, y de empleador a empleado.

Las relaciones entre los actores de una red social se generan a partir de procesos estocásticos. Por tanto, una parte considerable de la investigación en el análisis de redes sociales se ha enfocado en tratar de descubrir las características generales de estas redes [43]. De esta manera, se ha demostrado que las redes sociales poseen características comunes de gran importancia para el análisis, que las diferencian de las redes aleatorias. Las redes aleatorias fueron ampliamente estudiadas por el famoso matemático Paul Erdös [20], por lo que sus propiedades principales son bien conocidas por la comunidad científica.

Una de las características esenciales de las redes sociales es que la cantidad de aristas es lineal respecto a la cantidad de nodos, lo que en teoría de grafos sería equivalente a decir que los grafos son dispersos [3]. La distribución del grado de los nodos es otra característica de gran importancia para el diseño de algoritmos. Las redes sociales generalmente tienen un conjunto pequeño de nodos con alto grado, y uno numeroso de nodos con bajo grado, lo que se conoce como distribución de grados que sigue una ley de potencias [43, 19]. Cuando una red cumple esta propiedad se denomina red libre de escala. Estas características permiten diseñar algoritmos más

eficientes adaptados específicamente a estas redes.

Por otra parte, la distancia entre pares de nodos de la red está relacionada con la facilidad y rapidez en que se difunde la información. En redes sociales se ha comprobado que las distancias entre pares de nodos son relativamente cortas. A este fenómeno se le denomina efecto de mundo pequeño [3]. El modelo de generación de redes aleatorias Watts-Strogatz [57] fue propuesto para intentar dar una explicación a la existencia de este fenómeno en redes tan diversas como las redes eléctricas, las redes neuronales, redes de actores televisivos, entre otras.

El coeficiente de agrupamiento es una medida de la proporción de tríos de actores conectados entre sí, respecto a los tríos no conectados. En redes sociales se ha observado que esta proporción es mucho más alta de lo esperado. Esto puede ser interpretado de la siguiente manera: dados dos actores de una red social, mientras más actores estén relacionados a ambos actores (vecinos en común), es mayor la probabilidad de que dicho par de actores se relacionen. Esta característica está muy relacionada con la existencia de comunidades, o grupos de nodos con mayor número de conexiones entre sí que con el resto de la red [3]. Por otra parte, se ha comprobado que los nodos con grados similares tienen tendencia a conectarse entre sí, lo que se conoce como homofilia o asortatividad [43, 19].

No todos los actores de una red social tienen la misma importancia. Para determinar la importancia de cada actor, y poder establecer quiénes son los más importantes, se han propuesto medidas de centralidad [21]. Las medidas de centralidad usualmente utilizan la información topológica de la red, es decir, las relaciones entre los actores, para dar a cada actor un valor de su importancia relativa. Una de las principales medidas propuestas es la intermediación [2], y es precisamente esta medida la que será estudiada en el presente trabajo.

1.3. La intermediación: enfoques para su cálculo

La intermediación es una medida de centralidad propuesta por Freeman en 1977 [21]. Esta cuantifica la frecuencia o el número de veces que un nodo actúa como un puente a lo largo del camino más corto entre otros dos nodos [53]. La idea intuitiva es que si se eligen dos nodos y uno de los caminos más cortos entre ellos al azar, entonces los nodos con mayor intermediación serán aquellos que aparezcan con mayor probabilidad dentro de este camino. Por tanto, los nodos con mayor intermediación serían aquellos capaces de controlar los flujos de información en una red social, asumiendo que la información siempre fluye de un nodo a otro por el camino más corto posible.

La intermediación ha sido utilizada ampliamente en otros campos de la ciencia. En [40] se empleó para la identificación de nodos sensibles en redes biológicas. De manera similar, puede ser utilizada en sistemas electrónicos de redes de comunicación [54], sistemas de redes de tránsito público [46], redes de distribución de gas, redes de saneamiento de aguas residuales, etc. En redes de interacción proteína-proteína, las proteínas esenciales pueden ser identificadas por su alto valor de intermediación [32, 61, 26]. Por otra parte, demostró ser útil en la identificación de actores clave en redes terroristas [35].

1.3.1. Definición

La intermediación de un nodo se expresa según la Ecuación 1.1.

$$C_B(v) = \sum_{\substack{s \neq v, t \neq v \\ s,t \in V}} \delta_{st}(v) \tag{1.1}$$

donde $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$.

Para calcular la intermediación de un nodo es necesario determinar previamente la cantidad de caminos mínimos que pasan por este nodo. El valor de $\sigma_{st}(v)$ puede obtenerse utilizando la Ecuación 1.2.

$$\sigma_{st}(v) = \begin{cases} 0 & \text{si } dist_{sv} + dist_{vt} \neq dist_{st} \\ \sigma_{sv} \cdot \sigma_{vt} & \text{en otro caso} \end{cases} \tag{1.2}$$

Por tanto, bastaría calcular los valores σ_{st} y $dist_{st}$ para todos los pares de nodos s y t, lo cual puede lograrse en grafos no pesados utilizando el algoritmo de búsqueda en anchura (*Breath First Search*; BFS). Este paso requeriría un tiempo de ejecución de $\mathcal{O}(n \cdot m)$. Luego sería necesario calcular los valores de intermediación, lo cual requiere realizar $\mathcal{O}(n^3)$ operaciones. En general, siguiendo los pasos expuestos anteriormente, se puede calcular la intermediación de todos los nodos realizando $\mathcal{O}(n \cdot m + n^3)$ operaciones.

1.3.2. Algoritmo de Brandes

En [7] Brandes propuso el primer algoritmo capaz de calcular la intermediación de una manera más rápida. Para ello definió la función de dependencia $\delta_{s\cdot}(v)$ entre el nodo s y el nodo v como se muestra en la Ecuación 1.3.

$$\delta_{s\cdot}(v) = \sum_{t \in V} \delta_{st}(v) \tag{1.3}$$

Conociendo los valores de dependencia, es posible calcular la intermediación de todos los nodos en $\mathcal{O}(n^2)$ utilizando la Ecuación 1.4.

$$C_B(v) = \sum_{s \in V} \delta_{s\cdot}(v) \tag{1.4}$$

Para dar una solución eficiente al problema, Brandes demostró una fórmula recursiva (Ecuación 1.5) para calcular la dependencia conociendo previamente el DAG de caminos mínimos.

$$\delta_{s\cdot}(v) = \sum_{w:v\in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_{s\cdot}(w)) \tag{1.5}$$

donde $\{w : v \in P_s(w)\}$ es el conjunto de todos los nodos w tales que v es un predecesor de w en algún camino mínimo desde s. Para determinar los valores de intermediación el algoritmo de Brandes realiza los siguientes pasos:

1. Calcular los DAG de caminos mínimos a partir de cada nodo (incluyendo la cantidad de caminos mínimos y la lista predecesores). El tiempo de ejecución es $\mathcal{O}(n \cdot m)$ como se mencionó anteriormente.

2. Calcular los valores de dependencia utilizando la Ecuación 1.5. El tiempo de ejecución es $\mathcal{O}(n \cdot m)$.

3. Determinar los valores de intermediación usando la Ecuación 1.4. El tiempo de ejecución es $\mathcal{O}(n^2)$.

Concretamente el tiempo de ejecución del algoritmo propuesto por Brandes es $\mathcal{O}(n \cdot m + n^2)$. En lo que sigue denominaremos **Brandes** al algoritmo propuesto por el mismo autor. En la actualidad, este sigue siendo el algoritmo más utilizado en aplicaciones reales, y el punto de comparación esencial para los nuevos algoritmos dinámicos que se proponen.

1.3.3. Algoritmos del estado del arte

En los últimos años han sido propuestos varios algoritmos para enfrentar el problema del cálculo de la intermediación en el ambiente dinámico, con el objetivo de lograr una mayor eficiencia que **Brandes**. Estos pueden ser agrupados según las técnicas fundamentales utilizadas para dar una mejor solución al problema en: basados en algoritmos para calcular caminos mínimos y basados en la estructura del grafo [31]. En este trabajo nos enfocamos en el cálculo de la intermediación de manera exacta. Por esta razón, los algoritmos que calculan una aproximación de la intermediación [49, 28] no serán tratados.

Basados en algoritmos para calcular caminos mínimos

Los algoritmos en este grupo se basan en extender soluciones para el problema subyacente al cálculo de la intermediación, el cálculo de los caminos mínimos entre todos los pares de nodos de un grafo. En general, esta tarea no es sencilla ya que se hace necesario calcular algunos parámetros adicionales tales como la cantidad de caminos mínimos y los valores de dependencia (si se usa **Brandes** como base).

El algoritmo propuesto en [25] calcula la intermediación en grafos donde se insertan aristas con el paso del tiempo. Para ello, mantiene explícitamente los árboles a lo ancho partiendo de cada uno de los nodos del grafo. Los resultados experimentales muestran mejoría respecto a **Brandes**, pero su eficiencia computacional no es mejor que este en el peor caso. Su complejidad espacial es de $\mathcal{O}(n^2 + n \cdot m)$, por lo que se necesitaría una gran cantidad de memoria para ejecutar este algoritmo sobre un grafo de decenas de miles de nodos y aristas. Este algoritmo fue extendido para el ambiente paralelo y distribuido exitosamente en [55].

El algoritmo propuesto en [33] también es incremental, y puede procesar grafos dirigidos. Consiste en una extensión del algoritmo propuesto en [48]. Su compleji-

dad temporal es equivalente a **Brandes**, aunque en la práctica se obtienen mejoras considerables. Su complejidad espacial es cuadrática, lo que lo hace inadecuado para aplicaciones en grafos de gran tamaño. Es posible paralelizarlo a través de memoria compartida, pero su distribución exitosa no parece tarea sencilla. Para actualizar los valores de intermediación, entre una iteración y otra se almacenan la distancia y la cantidad de caminos mínimos entre todos los pares de nodos, y las listas de predecesores. Realizando el análisis de complejidad respecto a la salida [23], si definimos como salida todos los parámetros mencionados anteriormente, este algoritmo es cuadrático.

Por último, en [41] se propone otro algoritmo incremental para el cálculo de la intermediación que logra una mejoría en el costo computacional respecto a **Brandes**. Este mantiene un **DAG** de caminos mínimos para cada nodo del grafo. El mismo grupo de investigadores propuso posteriomente [42, 44] un algoritmo dinámico para el cálculo de la intermediación que tiene un gran valor teórico ya que logró de manera eficiente generalizar el algoritmo de [15], uno de los más eficientes para el cálculo de los caminos mínimos entre todos los pares de nodos en un grafo dinámico en la actualidad [16]. Sin embargo, estos trabajos no exponen resultados de experimentación, y las estructuras de datos utilizadas son complejas, por lo que su utilización práctica requiere alguna implementación eficiente comparable con el resto de las propuestas.

En [34] se extendió el algoritmo de Green [25] para el ambiente distribuido. Además, se desarrolló una implementación basada en Hadoop [1]. La complejidad espacial de este algoritmo, tal y como la de los propuestos en trabajos anteriores, es cuadrática, lo que es demasiado costoso para procesar grafos de gran tamaño.

Recientemente fue propuesto un nuevo algoritmo [5] (**Ibet**) para el cálculo de la intermediación en grafos dirigidos donde se insertan aristas. Este necesita almacenar las distancias entre todos los pares de nodos, por lo que su complejidad espacial es cuadrática, aunque mucho menor que los trabajos anteriores ya que solo almacena las

[1] https://hadoop.apache.org/ visitado el 23 de mayo de 2019

distancias entre los pares de nodos. Se basa en una propiedad de los caminos mínimos aplicada por primera vez al problema del cálculo de la intermediación, que permite calcular eficientemente la distancia y la cantidad de caminos mínimos entre todos los pares de nodos. Los experimentos realizados demostraron que esta propuesta supera a las anteriores de manera contundente.

Basados en la estructura del grafo

Todos los algoritmos reseñados anteriormente tienen en común que su complejidad espacial es cuadrática, lo que les permite ser mucho más eficientes en la práctica, pero no procesar grafos realmente grandes. Algunos algoritmos se han propuesto para resolver este problema, basándose en la descomposición de los grafos en estructuras que permitan calcular la intermediación de manera más sencilla y eficiente [37, 36, 31]. Usualmente estos algoritmos tienen una complejidad espacial lineal.

El primero de estos algoritmos fue **QuBE** [37], que determina los valores de intermediación en grafos dinámicos no dirigidos sin calcular nuevamente todos los caminos mínimos. Para ello, descompone el grafo en MUC[2] y actualiza la partición cuando se inserta o se elimina una arista. Además, calcula la intermediación al ocurrir una de estas modificaciones en el grafo de manera eficiente ejecutando **Brandes** solo para un subgrafo (que sería el MUC afectado por la modificación), y actualizando los demás valores sin necesidad de calcular los caminos mínimos. Por tanto, si el MUC afectado tiene un tamaño relativamente grande, el desempeño de este algoritmo sería equivalente a **Brandes**. A pesar de demostrar experimentalmente una mejoría notable en cuanto al tiempo de ejecución respecto a **Brandes**, la complejidad asintótica del algoritmo con respecto a la salida no está acotada, ya que la cantidad de operaciones que realiza no depende del tamaño de la modificación ni de la cantidad de nodos actualizados. En [36] este método fue refinado. Para ello, se utilizó la des-

[2]MUC [37]: Unión de ciclos mínimos, es la unión de ciertos ciclos en el grafo que forman una base algebraica y la suma de sus pesos es mínima.

composición en componentes biconexos, obteniéndose mejores resultados, dado que esta descomposición es más eficiente de calcular, y los componentes resultantes de menor tamaño.

Concurrentemente con [36], en [31] se propuso un algoritmo dinámico (**Icentral**) para el cálculo de la intermediación en grafos dinámicos no dirigidos y conexos. Entre una iteración y otra se almacena solamente la intermediación de todos los nodos, por lo que su complejidad espacial es lineal. Similar a **QuBE**, **Icentral** descompone el grafo en componentes biconexos, utilizando un algoritmo lineal, y luego actualiza la intermediación de los nodos en el componente biconexo donde la arista modificada reside, es decir, el componente afectado. Esto es suficiente, ya que como se demuestra en el artículo, para grafos no dirigidos, luego de una modificación, la intermediación de los nodos que no pertenecen al componente afectado se mantiene constante. La actualización de los nodos dentro del componente afectado se realiza de manera similar a **Brandes**, pero procesando solo aquellos nodos que, siendo origen de algún camino mínimo modificado, provoquen cambios en los valores de intermediación de algún otro nodo. Además, el algoritmo fue implementado en el ambiente paralelo con memoria compartida, y los experimentos realizados mostraron que tiene un mejor desempeño que el resto de las propuestas anteriores. Aún así, su eficiencia está directamente relacionada con el tamaño del componente afectado, ya que su complejidad temporal es cuadrática respecto al tamaño de dicho componente.

1.4. Consideraciones parciales

La intermediación es una de las medidas de centralidad más importantes en el Análisis de Redes Sociales. Su cálculo tiene una alta complejidad computacional, por lo que la optimización de esta tarea algorítmica ha sido ampliamente estudiada.

El algoritmo **Brandes** es reconocido como el de mejor desempeño en el ámbito estático (existe evidencia de que un algoritmo más eficiente no será posible [58]), y

sirve de base para todas las propuestas para el cálculo de la intermediación en el ámbito dinámico.

Los algoritmos dinámicos para el cálculo de la intermediación pueden dividirse en dos grupos, dependiendo de las técnicas algorítmicas utilizadas. En el primer grupo, los basados en optimizar el cálculo de los caminos mínimos, se encuentran los de menor tiempo de ejecución, pero todos con una complejidad espacial cuadrática, que imposibilita su uso en grafos de gran tamaño. En el segundo grupo, los basados en la estructura del grafo, se incluyen los que mejoran la eficiencia temporal de los algoritmos estáticos, y su complejidad espacial es lineal. Como detrimento, estos algoritmos no pueden procesar grafos dirigidos.

En la presente tesis proponemos el algoritmo **Bidcentral**, que viene a llenar este nicho faltante, ya que se clasifica en el segundo grupo, y puede procesar grafos dinámicos dirigidos.

Capítulo 2

Bidcentral: Algoritmo dinámico para calcular la intermediación

En este capítulo se describe el algoritmo dinámico **Bidcentral** para calcular la intermediación propuesto en la presente tesis. Además, se presenta un análisis de la correctitud y la complejidad computacional, así como elementos fundamentales de su implementación paralela.

2.1. Descripción general

Por simplicidad, en este trabajo nos restringiremos a grafos no pesados (dado que facilita notablemente la formalización) y simples (sin aristas múltiples ni lazos). El algoritmo propuesto es una generalización de **Icentral** para el procesamiento de grafos dirigidos.

Proponemos la definición que se muestra a continuación como una de las bases fundamentales de **Bidcentral**, ya que permite limitar los cálculos necesarios a un

conjunto de nodos de menor tamaño.

Definición 17. Sea G un grafo, y sea G^* el grafo que se obtiene luego de modificar una arista (insertar o eliminar) en G. Definimos como *componente afectado* al componente biconexo de la versión no dirigida de G^* (en el caso de la inserción) o de G (en el caso de la eliminación) al cual pertenece la arista modificada.

El principal obstáculo al generalizar **Icentral** es que, en grafos dirigidos, cuando una arista es modificada, los valores de intermediación de los nodos que no pertenecen al componente afectado pueden cambiar. Esto no ocurre en los grafos no dirigidos, como se demostró en el Teorema 1 en [31].

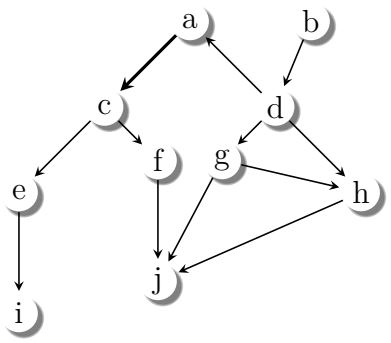

Figura 2.1: Grafo de ejemplo donde se inserta la arista de a a c.

En la Figura 2.1 mostramos un grafo con 10 nodos donde se inserta la arista de a a c. En este caso el componente afectado es el grafo inducido por el conjunto de nodos $\{a, c, d, f, g, h, j\}$. Claramente el nodo e no pertenece al componente afectado, pero el valor de su intermediación varía, ya que en el grafo inicial es 1 (por el camino mínimo de c a i), y en el final 4 (se agregan los caminos mínimos desde a, d y b). Esto hace que el Teorema 1 en [31] no sea aplicable para el caso de los grafos dirigidos. Como se explicará posteriormente, **Bidcentral** calcula estos cambios de

intermediación en los nodos externos al componente afectado de manera eficiente, basándose en un resultado propio del presente trabajo.

A continuación presentamos algunas definiciones y notaciones necesarias para comprender cómo **Bidcentral** actualiza los valores de intermediación cuando el grafo sufre modificaciones.

Definición 18. Se dice que un nodo v es una fuente afectada cuando, para algún nodo w del grafo, el valor de $\delta_{v\cdot}(w)$ (véase la Ecuación 1.3) se modifica.

Definición 19. Sea x un parámetro cualquiera del grafo o de alguno de sus nodos, entonces x^* representa el parámetro luego de que el grafo fue modificado. De la misma manera, x^r representa el parámetro calculado para el grafo con todas las aristas invertidas.

Definición 20. Sean a y b dos nodos, entonces denotaremos $a \to b$ si existe un camino desde a hasta b.

Definición 21. Sea B un componente biconexo, y $v \in B$ un punto de articulación del grafo, entonces $L(B, v)$ es el subgrafo inducido por todos los nodos w, tal que existe un camino de v a w o viceversa, y ninguna de las aristas contenida en este camino pertenece a B.

Definición 22. Sea A el componente afectado y $s \in A$ un punto de articulación, entonces $\text{reach}_o(s)$ es la cantidad de nodos v tal que $v \in L(A, s)$ y $v \to s$.

Definición 23. Sea A el componente afectado y $s \in A$ un punto de articulación, entonces $\text{reach}(s)$ es la cantidad de nodos v tal que existe un camino mínimo de v a s, y la última arista de este camino está contenida en A.

Bidcentral parte de los valores de intermediación de cada uno de los nodos (que pueden ser obtenidos a partir de cualquier algoritmo para el cálculo de la intermediación, por ejemplo **Brandes**). Luego de cada modificación en el grafo, actualiza estos valores. Para ello, cuenta con tres momentos fundamentales:

1. Calcular el componente afectado A.

2. Actualizar la intermediación de los nodos en A.

3. Actualizar la intermediación del resto de los nodos.

El pseudocódigo correspondiente a **Bidcentral** se muestra en el Algoritmo 1. Este comprende los momentos fundamentales del algoritmo mencionados anteriormente. El componente biconexo afectado puede ser calculado (línea 1) utilizando el conocido algoritmo propuesto en [30].

Algoritmo 1 actualizar-intermediación

Entrada: G, (u, v), C_B
Salida: C_B

1: $A \leftarrow$ componente-biconexo-afectado$(G, (u, v))$
2: $A^* \leftarrow A$ con arista (u, v) modificada (insertada o eliminada)
3: actualizar-intermediación-interno$(G, A, (u, v), A^*, C_B)$
4: actualizar-intermediación-externo(G, A, A^*, C_B)

La función *actualizar-intermediación-interno* que se muestra en el Algoritmo 2 se encarga de actualizar los valores de intermediación dentro del componente afectado previamente calculado. Los nodos que son fuentes afectadas se calculan en la línea 1 (a través del Algoritmo 3), determinando así para qué nodos s se pueden modificar los valores de $\delta_{s\cdot}$ (definido en la Ecuación 1.5). Luego, se determinan los puntos de articulación dentro del componente afectado (línea 2), que pueden ser obtenidos como subproducto del algoritmo para determinar los componentes biconexos.

Para el cálculo de la intermediación dentro del componente afectado se necesita saber cuántos nodos son accesibles (existe un camino hacia estos nodos) fuera del componente, y cuántos nodos externos pueden llegar a este. Estos parámetros se calculan en las líneas 3-6, y ya fueron definidos anteriormente.

Algoritmo 2 actualizar-intermediación-interno

Entrada: G, A, (u, v), A^*, C_B
Salida: C_B

1: $FA \leftarrow$ fuentes-afectadas$(A, (u, v))$
2: $PA \leftarrow$ puntos-articulación(G, A)
3: **Para Todo** $s \in PA$ **Hacer**
4: $\quad \text{reach}_o(s) \leftarrow |\{v \mid v \in L(A, s), v \to s\}|$
5: $\quad \text{reach}_o^r(s) \leftarrow |\{v \mid v \in L(A, s), s \to v\}|$
6: **Fin Para**
7: **Para Todo** $s \in FA$ **Hacer**
8: $\quad \delta_{s\cdot} \leftarrow$ Brandes-componente(A, s, reach_o^r)
9: $\quad \delta_{s\cdot}^* \leftarrow$ Brandes-componente$(A^*, s, \text{reach}_o^r)$
10: $\quad$ **Para Todo** $x \in A$ **Hacer**
11: $\quad\quad C_B(x) \leftarrow C_B(x) + \delta_{s\cdot}^*(x) - \delta_{s\cdot}(x)$
12: $\quad$ **Fin Para**
13: $\quad$ **Si** $s \in PA$ **Entonces**
14: $\quad\quad$ **Para Todo** $x \in A$ **Hacer**
15: $\quad\quad\quad C_B(x) \leftarrow C_B(x) + (\delta_{s\cdot}^*(x) - \delta_{s\cdot}(x)) \cdot \text{reach}_o(s)$
16: $\quad\quad$ **Fin Para**
17: $\quad$ **Fin Si**
18: **Fin Para**

Algoritmo 3 fuentes-afectadas

Entrada: A, (u, v)
Salida: FA

1: $FA \leftarrow$ lista vacía
2: **Para Todo** $s \in A$ **Hacer**
3: $\quad$ **Si** $dist_{su} + 1 \leq dist_{sv}$ **Entonces**
4: $\quad\quad$ Agregar s a FA
5: $\quad$ **Fin Si**
6: **Fin Para**

En las líneas 7-18 se actualizan los valores de intermediación de los nodos del componente afectado, a partir de las fuentes afectadas calculadas anteriormente. Para ello, primero es necesario calcular los valores de $\delta_{s\cdot}$ antes y después de la modificación (líneas 8 y 9). Esto se logra con la función *Brandes-componente* (Algoritmo 4), que reutiliza los valores previamente calculados de reach_o^r. Esta función procesa los nodos dentro del componente afectado A, y determina los valores de $\delta_{s\cdot}$ correspondientes. Para ello, utiliza un procedimiento muy similar a **Brandes**, con la diferencia de que cuando procesa un nodo w que es punto de articulación del grafo original (líneas 6-8), las contribuciones de los nodos en $L(A, w)$ al valor de $\delta_{s\cdot}(w)$ se calculan de manera inmediata, sin necesidad de procesar esos nodos. La validez de este procedimiento se demostrará en el siguiente epígrafe de esta tesis.

En las líneas 10-17 se actualizan propiamente los valores de intermediación de los nodos. Notemos que en el caso de que s sea un punto de articulación, son necesarias las líneas 13-17 ya que los nodos v fuera de A tal que $v \to s$ también contribuyen a la intermediación de los nodos de A. Con estas operaciones culmina la actualización de la intermediación en el componente afectado.

La función *actualizar-intermediación-externo* (Algoritmo 5) determina, en un primer momento, los puntos de articulación (estos ya habían sido calculados previamente, así que estos valores se pueden reutilizar). Luego, en las líneas 2-15, se actualizan los valores de la intermediación de todos los nodos que pertenecen a $L(A, s)$, para cada punto de articulación s del componente afectado.

Actualizar la intermediación de un nodo externo x implica calcular los valores de $\delta_{s\cdot}(x)$ correspondientes (líneas 5 y 11), así como la cantidad de nodos cuyos caminos mínimos hacia x pasan por s (líneas 3 y 4), y la cantidad de nodos cuyos caminos mínimos desde x pasan por s (líneas 9 y 10), antes y después de que el grafo haya sido modificado. Para el cálculo de $\delta_{s\cdot}(x)$ se utiliza la función *Brandes-similar* (Algoritmo 6), muy similar a **Brandes** pero restringida al subgrafo $L(A, s)$. Para calcular las distintas variantes de reach (como en la Definición 23) se utiliza la función *BFS-interno* (Algoritmo 7), que no es más que un BFS clásico para determinar

Algoritmo 4 Brandes-componente

Entrada: A, s, reach_o^r
Salida: δ

// Calcular δ dentro del componente afectado
$(\sigma, P, \text{visitados}) \leftarrow BFS(A, s)$
Para Todo v en $L(A, s)$ **Hacer**
    $\delta(v) \leftarrow 0$
Fin Para
Para Todo w en visitados invertido **Hacer**
    Si $w \neq s$ y $w \in PA$ **Entonces**
        $\delta(w) \leftarrow \delta(w) + \text{reach}_o^r(w)$
    Fin Si
    Para Todo $p \in P(w)$ **Hacer**
        $\delta(p) \leftarrow \delta(p) + \frac{\sigma(p)}{\sigma(w)}(1 + \delta(w))$
    Fin Para
Fin Para

Algoritmo 5 actualizar-intermediación-externo

Entrada: G, A, A^*, C_B
Salida: C_B

$PA \leftarrow$ puntos-articulación(G, A)
Para Todo $s \in PA$ **Hacer**
    $\text{reach}(s) \leftarrow \text{BFS-interno}(A, s, \text{reach}_o^r)$
    $\text{reach}^*(s) \leftarrow \text{BFS-interno}(A^*, s, \text{reach}_o^r)$
    $\delta_{s\cdot} \leftarrow \text{Brandes-similar}(L(A, s), s)$
    Para Todo $x \in \{v \mid v \in L(A, s), s \rightarrow v\}$ **Hacer**
        $C_B(x) \leftarrow C_B(x) + \delta_{s\cdot}(x) \cdot (\text{reach}^*(s) - \text{reach}(s))$
    Fin Para
    $\text{reach}^r(s) \leftarrow \text{BFS-interno}(A^r, s, \text{reach}_o)$
    $\text{reach}^{*r}(s) \leftarrow \text{BFS-interno}(A^{*r}, s, \text{reach}_o)$
    $\delta_{s\cdot}^r \leftarrow \text{Brandes-similar}(L(A, s)^r, s)$
    Para Todo $x \in \{v \mid v \in L(A, s), v \rightarrow s\}$ **Hacer**
        $C_B(x) \leftarrow C_B(x) + \delta_{s\cdot}^r(x) \cdot (\text{reach}^{*r}(s) - \text{reach}^r(s))$
    Fin Para
Fin Para

cuántos nodos son alcanzables a partir de un nodo en un grafo, pero más eficiente porque reduce los cálculos realizados ya que reutiliza los valores de reach_o calculados anteriormente para todos los puntos de articulación del componente afectado. Los valores de intermediación son actualizados propiamente en las líneas 6-8 y 12-14. Como se ha expresado anteriormente, la validez de este procedimiento se demostrará en el siguiente epígrafe.

Algoritmo 6 Brandes-similar

Entrada: $L(A, s)$, s
Salida: δ

```
    // Calcular δ fuera de componente afectado
1:  (σ, P, visitados) ← BFS(L(A, s), s)
2:  Para Todo v ∈ L(A, s) Hacer
3:      δ(v) ← 0
4:  Fin Para
5:  Para Todo w en visitados invertido Hacer
6:      Para Todo p ∈ P(w) Hacer
7:          δ(p) ← δ(p) + (σ(p)/σ(w))(1 + δ(w))
8:      Fin Para
9:  Fin Para
```

Algoritmo 7 BFS-interno

Entrada: H, s, reach

Salida: cant

// BFS modificado para calcular la cantidad de nodos alcanzables desde un punto de articulación del componente afectado

```
1:  visitados ← cola de nodos vacía
2:  cant ← 0
3:  Agregar s a visitados
4:  Para Todo v en visitados Hacer
5:      cant ← cant + 1
6:      Para Todo nv vecino de v en H Hacer
7:          Si nv no está contenido en visitados  Entonces
8:              Agregar nv a visitados
9:              Si nv ∈ PA Entonces
10:                 cant ← cant + reach(nv)
11:             Fin Si
12:         Fin Si
13:     Fin Para
14: Fin Para
```

Algoritmo 8 BFS

Entrada: H, s

Salida: $\sigma, P,$ visitados

```
    // BFS aumentado para calcular cantidad de caminos mínimos, listado de predecesores, y los nodos visitados en orden de profundidad
 1: visitados ← cola de nodos vacía
 2: Para Todo v ∈ H Hacer
 3:     d(v) ← 0 // distancia
 4:     σ(v) ← 0
 5:     P(v) ← lista vacía
 6: Fin Para
 7: Agregar s a visitados
 8: Para Todo v in visitados Hacer
 9:     Para Todo nv vecino de v en H Hacer
10:         Si nv no está contenido en visitados  Entonces
11:             d(nv) ← d(v) + 1
12:             σ(nv) ← σ(v)
13:             Agregar nv a visitados
14:             Agregar v a P(nv)
15:         Si No
16:             Si d(nv) = d(v) + 1 Entonces
17:                 σ(nv) ← σ(nv) + σ(v)
18:                 Agregar v a P(nv)
19:             Fin Si
20:         Fin Si
21:     Fin Para
22: Fin Para
```

2.2. Correctitud

En este epígrafe demostraremos que el algoritmo **Bidcentral** calcula de manera correcta la variación de la intermediación de todos los nodos de un grafo cuando este sufre modificaciones. Para ello, son necesarias algunas propiedades relativas a la variación de la intermediación de los nodos de un grafo dinámico.

En el Teorema 1 demostramos una fórmula que nos permite computar la variación de la intermediación que sufre un nodo externo al componente afectado de manera eficiente. Este es el principal resultado de la tesis, ya que permite obtener estos valores de manera eficiente, y constituyó la base fundamental para generalizar **Icentral**.

Teorema 1. Sea x un nodo que no pertenece al componente afectado A, y sea s el punto de articulación dentro del componente tal que su eliminación desconecta x de A. Luego de una modificación en el grafo, la variación de la intermediación de x se calcula según la Ecuación 2.1.

$$\delta_{s\cdot}(x) \cdot (\text{reach}^*(s) - \text{reach}(s)) + \delta^r_{s\cdot}(x) \cdot (\text{reach}^{*r}(s) - \text{reach}^r(s)) \tag{2.1}$$

Demostración. Para ganar en claridad, renombraremos las variables en la definición de intermediación de la Ecuación 1.4 por las que se presentan en la Ecuación 2.2.

$$C_B(x) = \sum_{\substack{a \neq x, b \neq x \\ a,b \in V}} \frac{\sigma_{ab}(x)}{\sigma_{ab}} \tag{2.2}$$

En la suma de la derecha de la Ecuación 2.2, los únicos términos que pueden cambiar luego de una modificación son aquellos tales que a y b están en componentes biconexos diferentes, y los caminos mínimos de a a b pasan por A. Esto ocurre ya que, si a y b pertenecen al mismo componente biconexo, y alguno de sus caminos mínimos pasa por x, entonces x también pertenece a ese componente biconexo (que

no es el componente afectado), y por tanto, ningún camino mínimo entre a y b resulta afectado por la modificación del grafo. De la misma manera, si a y b no están en el mismo componente biconexo, y algún camino mínimo de a a b no pasa por A, entonces ningún camino lo hace, y por tanto, ninguno resultaría afectado.

Luego, todos los caminos mínimos de a a b deben pasar por s. El orden relativo de s y x en los caminos de a a b que pasan por x determina dos casos posibles:

1. $a, s, x, b \implies \frac{\sigma_{ab}(x)}{\sigma_{ab}} = \frac{\sigma_{as}\sigma_{sx}\sigma_{xb}}{\sigma_{as}\sigma_{sb}} = \frac{\sigma_{sx}\sigma_{xb}}{\sigma_{sb}} = \frac{\sigma_{sb}(x)}{\sigma_{sb}}$
2. $a, x, s, b \implies \frac{\sigma_{ab}(x)}{\sigma_{ab}} = \frac{\sigma_{ax}\sigma_{xs}\sigma_{sb}}{\sigma_{as}\sigma_{sb}} = \frac{\sigma_{ax}\sigma_{xs}}{\sigma_{as}} = \frac{\sigma_{as}(x)}{\sigma_{as}}$

Luego, los términos que pueden cambiar son iguales a:

$$\sum_{a,b \text{ en caso } 1} \frac{\sigma_{sb}(x)}{\sigma_{sb}} + \sum_{a,b \text{ en caso } 2} \frac{\sigma_{as}(x)}{\sigma_{as}} = \text{reach}(s) \cdot \delta_{s\cdot}(x) + \text{reach}^r(s) \cdot \delta^r_{s\cdot}(x) \qquad (2.3)$$

Notemos que los valores de $\delta_{s\cdot}(x)$ y $\delta^r_{s\cdot}(x)$ no varían cuando el grafo es modificado, ya que todos los caminos mínimos de s a x, y viceversa, están fuera del componente biconexo afectado. A partir de este resultado se deduce de manera sencilla la afirmación del teorema.

□

Los teoremas que proponemos a continuación son la base fundamental del cálculo de la intermediación en el interior del componente afectado.

Teorema 2. Supongamos que en un grafo G se inserta una arista. Supongamos además, que para un nodo s existe un nodo x tal que el valor de $\delta_{s\cdot}(x)$ cambia luego de la modificación del grafo. En el grafo modificado existe un camino mínimo de s a algún otro nodo del grafo que pasa por la arista insertada.

Demostración. Como $\delta_{s\cdot}(x)$ cambia y es igual a $\sum_{w \in V} \frac{\sigma_{sw}(x)}{\sigma_{sw}}$, entonces para algún w el valor de $\sigma_{sw}(x)$ o de σ_{sw} cambia. Esto solo ocurre cuando la arista insertada crea un nuevo camino mínimo entre s y w, que puede o no pasar por x. Luego, en el grafo modificado, existe al menos un camino mínimo que parte de s, y pasa por la arista insertada, como se quería demostrar. □

Corolario 2. Teniendo en cuenta el teorema anterior, podemos decir que si la arista insertada es (u, v), entonces el nodo s pertenece al conjunto de nodos fuentes afectados si y solo si $dist_{su} + 1 \leq dist_{sv}$, donde estas distancias se calculan en el grafo original.

Teorema 3. Supongamos que en un grafo G se elimina una arista. Supongamos además, que para un nodo s existe un nodo x tal que el valor de $\delta_{s\cdot}(x)$ cambia luego de la modificación del grafo. En el grafo original existe un camino mínimo de s que pasa por la arista eliminada.

Demostración. Como $\delta_{s\cdot}(x)$ cambia y es igual a $\sum_{w \in V} \frac{\sigma_{sw}(x)}{\sigma_{sw}}$, entonces para algún w el valor de $\sigma_{sw}(x)$ o de σ_{sw} cambia. Esto solo ocurre cuando la arista eliminada era parte de un camino mínimo entre s y w, que puede o no pasar por x. Luego, en el grafo original existe al menos un camino mínimo que parte de s, y pasa por la arista eliminada, como se quería demostrar. □

Corolario 3. Teniendo en cuenta el teorema anterior, podemos decir que si la arista eliminada es (u, v), entonces el nodo $s \in A$ pertenece al conjunto de nodos afectados si y solo si $dist_{su} + 1 = dist_{sv}$, donde estas distancias se calculan en el grafo original. Notemos que como en el grafo original existe la arista (u, v), esto es equivalente a $dist_{su} + 1 \leq dist_{sv}$.

Teorema 4. Supongamos que v es un nodo del componente biconexo afectado A y $s \in A$ es un punto de articulación, entonces la contribución de los nodos en $L(A, s)$ a $\delta_{v\cdot}(s)$ es igual a $\text{reach}_o^r(s)$.

Demostración. Por definición, la contribución de los nodos en $L(A, s)$ a $\delta_{v\cdot}(s)$ es igual a $\sum_{w \in L(A,s)} \frac{\sigma_{vw}(s)}{\sigma_{vw}}$. Pero $\forall w \in L(A, s)$, tal que $s \to w$, se cumple que $\sigma_{vw}(s) = \sigma_{vw}$,

ya que todos los caminos entre v y w pasan por s. Por tanto,

$$\sum_{w \in L(A,s)} \frac{\sigma_{vw}(s)}{\sigma_{vw}} = \sum_{\substack{w \in L(A,s) \\ s \to w}} 1 = \text{reach}_o^r(s)$$

como se quería demostrar. □

Teorema 5. Sea s un punto de articulación en el componente biconexo afectado A y $v \in L(A, s)$ un nodo tal que $v \to s$, entonces para todo nodo $x \in A$ tal que $\delta_{v\cdot}(x)$ cambia cuando el grafo sufre una modificación, se cumple que $\delta_{v\cdot}(x) = \delta_{s\cdot}(x)$ y $\delta^*_{v\cdot}(x) = \delta^*_{s\cdot}(x)$.

Demostración. Notemos que:

$$\delta_{v\cdot}(x) = \sum_{w \in L(A,s)} \frac{\sigma_{vw}(x)}{\sigma_{vw}} + \sum_{w \notin L(A,s)} \frac{\sigma_{vw}(x)}{\sigma_{vw}} = \sum_{w \notin L(A,s)} \frac{\sigma_{vw}(x)}{\sigma_{vw}}$$

ya que ningún camino entre v y un nodo en $L(A, s)$ pasa por x, dado que $x \in A$. Luego,

$$\sum_{w \notin L(A,s)} \frac{\sigma_{vw}(x)}{\sigma_{vw}} = \sum_{w \notin L(A,s)} \frac{\sigma_{vs}\sigma_{sw}(x)}{\sigma_{vs}\sigma_{sw}} = \sum_{w \notin L(A,s)} \frac{\sigma_{sw}(x)}{\sigma_{sw}} = \delta_{s\cdot}(x)$$

como se quería demostrar. La deducción para $\delta^*_{v\cdot}(x) = \delta^*_{s\cdot}(x)$ es equivalente. □

Ahora pasaremos a demostrar propiamente que el algoritmo **Bidcentral** actualiza de manera correcta los valores de intermediación de todos los nodos de un grafo cuando este sufre alguna modificación, ya sea la inserción o eliminación de una arista. Para ello, basta demostrar que la función *actualizar-intermediación-interno* actualiza los valores en los nodos dentro del componente afectado de manera correcta, y que la función *actualizar-intermediación-externo* hace lo mismo en los nodos fuera de dicho componente.

Primero analizaremos la función *actualizar-intermediación-interno*. Los nodos fuentes afectados dentro del componente afectado son calculados de manera correcta por la función *fuentes-afectadas* según Colorario 2 y Colorario 3. Por definición, estos son los únicos nodos dentro del componente afectado que pueden contribuir a variaciones en la intermediación. Por tanto, solo para estos nodos es necesario calcular los valores de $\delta_{s\cdot}$ (líneas 8 y 9), a través de la función *Brandes-componente*. Esta función calcula los valores requeridos de una manera muy similar a **Brandes**. Sin embargo, no necesita procesar los nodos fuera de A, debido al Teorema 4.

Luego, la contribución de los nodos dentro de A a la intermediación de los nodos dentro de A queda correctamente actualizada en las líneas 10-12. Solo quedaría agregar la contribución de los nodos fuera de A, y esto ocurre en las líneas 13-17. Todo nodo externo a A pertenece a $L(A, s)$, para algún punto de articulación s. Teniendo en cuenta el Teorema 5, la contribución de todos los nodos en $L(A, s)$ es la misma (igual a $\delta_{s\cdot}(x)$), y la cantidad de estos nodos es $\text{reach}_o(s)$, por lo que la línea 15 actualiza los valores correspondientes de manera correcta.

Ahora analizaremos la función *actualizar-intermediación-externo*. Recordemos que esta función procesa los nodos externos al componente afectado, agrupados según su pertenencia a $L(A, s)$, donde s es un punto de articulación. En las líneas 3-4 y 9-10 se calculan las variantes de los valores de reach, utilizando la función *BFS-interno*. Esta función determina la cantidad de nodos que pueden ser alcanzados a partir de un punto de articulación. De la misma forma que la función *Brandes-componente*, esta no necesita procesar nodos fuera del componente afectado, ya que puede reutilizar los cálculos almacenados en los valores de reach_o^r y reach_o. Notemos que estos valores son los mismos para el grafo modificado, ya que son independientes de las aristas en el componente afectado.

Por otra parte, la función *Brandes-similar*, que calcula los valores de $\delta_{s\cdot}$, es equivalente a **Brandes**, restringido al subgrafo $L(A, s)$. Luego, por el Teorema 1, las líneas 6-8 y 12-14 (en la función *actualizar-intermediación-externo*) actualizan de manera correcta los valores de intermediación de todos los nodos externos a A.

2.3. Análisis de complejidad

En este epígrafe demostraremos que la complejidad espacial y la complejidad temporal de **Bidcentral** son equivalentes a las de **Icentral**, a pesar de que permite procesar grafos dirigidos y no necesariamente conexos.

2.3.1. Complejidad espacial

Bidcentral actualiza la intermediación de todos los nodos de un grafo dinámico a través de la función *actualizar-intermediación*. Calcular el componente biconexo afectado A, y su versión modificada A^*, requiere una cantidad de espacio proporcional al tamaño del grafo (líneas 1 y 2).

Actualizar los valores de intermediación dentro del componente afectado es el objetivo de la función *actualizar-intermediación-interno*. Las fuentes afectadas (línea 1), los puntos de articulación (línea 2), y los valores de reach (líneas 4 y 5), requieren un espacio proporcional a la cantidad de nodos del grafo. Los valores de $\delta_{s\cdot}$ (líneas 8 y 8), para un s fijo, no son necesarios en otras iteraciones del ciclo (líneas 7-18), por lo que pueden ser almacenados utilizando un espacio proporcional a la cantidad de nodos del grafo. Lo mismo ocurre con las estructuras internas utilizadas en la función *Brandes-componente*.

Por último, la función *actualizar-intermediación-externo* actualiza los valores de intermediación fuera del componente afectado. Teniendo en cuenta que, para cada punto de articulación s, los valores de reach(s) y $\delta_{s\cdot}$ son independientes en cada iteración, el almacenamiento necesario es proporcional a la cantidad de nodos del grafo. Por otra parte, la función *BFS-interno* procesa los nodos dentro de A, por lo que su complejidad espacial es lineal respecto a este subgrafo.

Resumiendo, la complejidad espacial de **Bidcentral** es lineal respecto a la can-

tidad de aristas y nodos del grafo. Solo C_B y el grafo en sí persisten a través de las actualizaciones. Esto es lo mejor que puede lograr un algoritmo dinámico para el cálculo de la intermediación, por lo que, en este sentido, su complejidad espacial es óptima.

2.3.2. Complejidad temporal

La complejidad temporal de **Bidcentral** es igual a la complejidad de hallar los componentes biconexos, más la complejidad de la función *actualizar-intermediación-interno*, más la de *actualizar-intermediación-externo*. Existen varios algoritmos con complejidad lineal para el cálculo de los componentes biconexos. Estos permiten determinar a qué componente biconexo pertenece cada arista del grafo, por lo que saber el componente biconexo afectado es trivial.

Analicemos ahora la función *actualizar-intermediación-interno*. Sean n_A y m_A el número de nodos y aristas en el componente afectado respectivamente, entonces la cantidad de fuentes afectadas es a lo sumo n_A, ya que todos estos nodos pertenecen a A. El tiempo necesario para calcular las fuentes afectadas y los puntos de articulación es claramente lineal en el tamaño del grafo.

En las líneas 3-6 se calcula la cantidad de nodos v en $L(A, s)$, donde s es un punto de articulación de A, tal que $v \to s$ o viceversa. Notemos que los subgrafos $L(A, s)$ son disjuntos, por lo que la cantidad de operaciones requeridas para determinar esos valores es lineal en el tamaño del grafo.

Las operaciones en las líneas 8 y 9 tienen una complejidad lineal en el tamaño de A, ya que realizan un recorrido muy parecido a los realizados en **Brandes**, restringidos en este caso al componente afectado, sin procesar ningún nodo externo a este. El resto de las operaciones que se realizan dentro del ciclo de las líneas 7-18 es constante, por lo que el tiempo de ejecución de esta parte puede medirse sin mucha

dificultad: $\mathcal{O}(|FA| \cdot (n_A + m_A))$.

Sumando los valores determinados anteriormente, el tiempo de ejecución de la función *actualizar-intermediación-interno* es $\mathcal{O}(|FA| \cdot (n_A + m_A) + n + m)$.

Analicemos ahora la función *actualizar-intermediación-externo*. Los puntos de articulación (línea 1) ya fueron calculados anteriormente, por lo que se pueden reutilizar esos valores. Cada una de las llamadas a la función *BFS-interno* es lineal en el tamaño de A, ya que esta función realiza un recorrido en A y no procesa ningún nodo externo.

Como se ha mencionado anteriormente, los subgrafos $L(A, s)$ con $s \in PA$ son disjuntos, por lo que la cantidad de operaciones requeridas para procesarlos con la función *Brandes-similar* es lineal en el tamaño del grafo. Lo mismo ocurre con los ciclos en las líneas 6-8 y 12-14, ya que las operaciones internas a los ciclos (líneas 7 y 13) son constantes.

Sumando los valores determinados anteriormente, el tiempo de ejecución de la función *actualizar-intermediación-externo* es $\mathcal{O}(|PA| \cdot (n_A + m_A) + n + m)$.

Teniendo en cuenta los análisis realizados anteriormente para actualizar la intermediación dentro y fuera del componente afectado, la complejidad temporal de **Bidcentral** es $\mathcal{O}((|FA| + |PA|) \cdot (n_A + m_A) + n + m)$. Esto es equivalente a $\mathcal{O}(n_A \cdot (n_A + m_A) + n + m)$ en el peor caso, ya que la cantidad de puntos de articulación y nodos fuentes afectados es a lo sumo la cantidad de nodos en A. Este resultado coincide con el alcanzado por **Icentral**.

2.4. Implementación paralela

Paralelizar la parte más costosa de **Bidcentral**, el cálculo de la intermediación dentro del componente afectado y de los valores de reach en la función *actualizar-intermediación-externo*, es relativamente sencillo. Como los valores δ respecto a los nodos fuente afectados son calculados independientemente, estos cálculos pueden ser realizados por diferentes procesadores en un ambiente paralelo.

Supongamos que se tiene una computadora con k procesadores. El procesamiento de los nodos fuente afectados (en la función *actualizar-intermediación-interno*) podría repartirse entre estos procesadores, y en cada uno sería necesario una capacidad de memoria lineal respecto al tamaño del grafo. La misma situación ocurre con los valores de reach en la función *actualizar-intermediación-externo*.

Luego, la complejidad espacial del algoritmo en su variante paralela sería $\mathcal{O}(n + m + k \cdot n_A)$, y su complejidad temporal $\mathcal{O}(n + m + n_A \cdot (n_A + m_A)/k)$ (en el mejor caso, cuando el tiempo de procesamiento en cada procesador es similar). En este contexto, se espera que el tiempo de ejecución se reduzca considerablemente, similar a los experimentos en [31].

2.5. Conclusiones parciales

El algoritmo **Bidcentral** permite calcular la intermediación en grafos dinámicos, dirigidos y no necesariamente conexos. Para ello, reduce la parte más costosa del procesamiento a un subgrafo, el componente biconexo afectado, utilizando la técnica de descomposición en componentes biconexos. Un resultado propio del presente trabajo (Teorema 1), específico para grafos dirigidos, permitió calcular de manera eficiente la intermediación de los nodos fuera de este componente.

Su complejidad espacial es lineal, un resultado óptimo para el problema en cuestión, teniendo en cuenta que es necesario almacenar el grafo. Su complejidad temporal es equivalente a la de **Icentral**, a pesar de procesar grafos dirigidos y no necesariamente conexos, y por tanto teniendo un carácter más general.

Por la naturaleza del algoritmo, se esperan mejoras considerables en tiempo en el ambiente paralelo, aunque requeriría un mayor almacenamiento.

Capítulo 3

Evaluación del algoritmo Bidcentral

En este capítulo validamos a partir de experimentos el desempeño del algoritmo propuesto **Bidcentral**. Para ello utilizamos grafos sintéticos y reales que representan redes sociales. Se implementaron con objetivos comparativos algoritmos recientes que han obtenido buenos resultados en la literatura, como **Brandes**, **Ibet** y **Icentral**.

3.1. Conjuntos de datos: grafos reales y sintéticos

Los conjuntos de datos utilizados para la experimentación fueron tomados de fuentes en Internet, algunos de ellos previamente referenciados en [5] o [31]; y otros tomados de la colección de grafos SNAP [1]. Las principales redes consideradas en el estudio se relacionan a continuación:

- *ego-Facebook*: red de círculos sociales (o listas de amigos) de Facebook, creada

[1] `http://snap.stanford.edu/data/index.html`

a partir de ciertos usuarios (ego).

- *ca-GrQc*: red de colaboración en el tema de la relatividad general, del archivo en línea ArXiv.
- *Epa* [2]: red de páginas web.
- *Eva* [3]: red de propietarios.
- *ca-HepTh*: red de colaboración en el tema de teoría de la física de alta energía, del archivo en línea ArXiv.
- *email-Eu-core*: red de correos electrónicos de una institución de investigación europea de gran tamaño.
- *CollegeMsg*: red de mensajes electrónicos privados dentro de la red social en línea en la Universidad de California, Irvine.
- *p2p-Gnutella08*, *p2p-Gnutella09*, *p2p-Gnutella05* y *p2p-Gnutella04*: redes punto a punto del proyecto *Gnutella* de distribución de archivos, colectadas en 2002.
- *Wiki-Vote*: red de votación de la selección de administradores de Wikipedia.
- *wiki-RfA*: red de peticiones para convertirse en administrador entre los editores de Wikipedia.

También se utilizaron grafos aleatorios generados siguiendo el modelo R-MAT[11] [4]. La descripción de los datos se muestra en la Tabla 3.1.

[2] `http://www.cs.cornell.edu/courses/cs685/2002fa/data/gr0.epa`
[3] `http://vlado.fmf.uni-lj.si/pub/networks/data/econ/Eva/Eva.htm`
[4] Generados utilizando la biblioteca snappy con parámetros: $A = 0.75$, $B = 0.1$, $C = 0.1$

Tabla 3.1: Estadísticas de los conjuntos de datos (lbc se refiere al tamaño del mayor componente biconexo)

Conjunto de Datos	# nodos	# aristas	diámetro	lbc	tipo de arista
RMAT-1024[4]	653	8180	6	495	dirigido
RMAT-2048[4]	1166	16371	6	927	dirigido
RMAT-3072[4]	1603	24564	7	1261	dirigido
RMAT-4096[4]	2229	32754	6	1723	dirigido
RMAT-5120[4]	2582	40939	6	2008	dirigido
RMAT-6144[4]	2877	49134	7	2229	dirigido
RMAT-7168[4]	3821	57329	6	2911	dirigido
RMAT-8192[4]	3740	65520	6	2833	dirigido
RMAT-9216[4]	4157	73709	6	3148	dirigido
RMAT-10240[4]	5078	81894	7	3893	dirigido
RMAT-11264[4]	5210	90088	9	3942	dirigido
RMAT-12288[4]	5722	98284	7	4340	dirigido
RMAT-13312[4]	5526	106472	7	4107	dirigido
RMAT-14336[4]	6139	114661	6	4606	dirigido
RMAT-15360[4]	6539	122853	7	4881	dirigido
RMAT-16384[4]	7265	131047	6	5422	dirigido
ego-Facebook[1]	4039	88234	8	3698	no dirigido
ca-GrQc[1]	4158	13428	16	2651	no dirigido
Epa[2]	4253	8897	10	2163	no dirigido
Eva[3]	4475	4654	17	234	no dirigido
ca-HepTh[1]	8638	24806	17	5898	no dirigido
email-Eu-core[1]	986	24929	7	891	dirigido
CollegeMsg[1]	1893	20292	8	1498	dirigido
p2p-Gnutella08[1]	6299	20776	9	4535	dirigido
Wiki-Vote[1]	7066	103663	7	4786	dirigido
p2p-Gnutella09[1]	8104	26008	10	5606	dirigido
p2p-Gnutella05[1]	8842	31837	9	6830	dirigido
p2p-Gnutella04[1]	10876	39994	9	8379	dirigido
wiki-RfA[1]	11381	188910	7	7726	dirigido

3.2. Diseño experimental

El algoritmo propuesto fue evaluado experimentalmente midiendo el tiempo de ejecución y la memoria máxima utilizada, comparando estos valores con los obtenidos por **Icentral**, **Ibet** y **Brandes**. **Brandes** fue incluido ya que es utilizado como algoritmo de base en cualquier comparación entre algoritmos para el cálculo de la intermediación. El algoritmo **Ibet** es el que ha mostrado mayor eficiencia temporal y espacial dentro del grupo de los basados en caminos mínimos, por lo que su inclusión en los experimentos es necesaria. El algoritmo propuesto **Bidcentral** es una generalización de **Icentral**, por lo que este último fue incluido para mostrar las diferencias y similitudes entre ambos desde el punto de vista experimental.

Todos los algoritmos fueron implementados en el lenguaje Python [5], y los grafos fueron almacenados y manipulados utilizando la biblioteca de Python NetworkX [6]. Para que todos los algoritmos tuvieran las mismas condiciones, en el caso de **Brandes** y **Icentral**, se utilizaron las variantes no paralelas. Las implementaciones fueron ejecutadas en una máquina con la distribución Arch de GNU/Linux de 64 bit, un procesador Intel(R) Core(TM) i7-7700 CPU @ 2.80GHz, y 10 GB de memoria principal.

3.3. Resultados y discusión

Se realizaron un total seis de experimentos, que se diferencian en el tipo de grafo, el tipo de modificación y los algoritmos utilizados para la comparación. En cada uno de estos se repite 100 veces lo siguiente: se inserta/elimina una arista aleatoriamente, y se mide el tiempo de ejecución y la memoria máxima utilizada que requiere cada uno de los algoritmos para actualizar la intermediación de todos los nodos del grafo.

[5] `https://www.python.org`
[6] `http://networkx.github.io`

Luego, se calcula la media y varianza del tiempo de ejecución, así como la máxima memoria utilizada por cada algoritmo para las 100 repeticiones.

Experimentos en grafos incrementales

En los primeros tres experimentos, se comparó **Bidcentral** en su faceta incremental con **Brandes**, **Ibet** e **Icentral**. Cada experimento se corresponde a un tipo de grafo particular: no dirigido real, dirigido real y dirigido sintético. Para ello, para cada grafo, se seleccionaron 100 aristas de manera aleatoria, que no estuvieran contenidas previamente en el grafo y se insertaron de manera independiente.

Los resultados para grafos no dirigidos reales se muestran en la Tabla 3.2. En este caso, para utilizar los algoritmos **Ibet** y **Bidcentral**, que trabajan con grafos dirigidos, se transformó cada arista no dirigida en un par de aristas, una para cada dirección.

Como se esperaba, **Icentral** y **Bidcentral** tuvieron una eficiencia similar, en tiempo y espacio, y son más eficientes que **Brandes** en la mayoría de los casos. Esta mejora es altamente dependiente del tamaño del componente afectado. El mejor desempeño respecto a **Brandes** se obtuvo en el conjunto de datos *Eva*, donde el número de nodos en el mayor componente biconexo es relativamente pequeño. Solo en el caso de *ego-Facebook*, **Brandes** fue más eficiente, ya que en este grafo el componente biconexo de mayor tamaño comprende casi todos los nodos del grafo. En promedio, **Bidcentral** fue tres veces más rápido que **Brandes**.

Por otra parte, **Ibet** es el más rápido en la mayoría de los conjuntos de datos, pero su uso de memoria es elevado, haciéndolo muy costoso para grafos con decenas de miles de nodos. Notemos también, que para conjuntos de datos como *Eva*, **Ibet** es sobrepasado por algoritmos como **Icentral** y **Bidcentral**, recalcando la relevancia de algoritmos que utilizan la descomposición en componentes biconexos.

Tabla 3.2: Resultados para los grafos reales no dirigidos en ambiente incremental. Tiempo dado en segundos (media μ y desviación estándar γ) y memoria en MBytes (m).

Datos	**Brandes**			**Ibet**		
	μ	γ	m	μ	γ	m
ego-Facebook	22.48	1.20	152	**7.18**	5.06	1688
ca-GrQc	19.05	0.28	126	**4.02**	0.81	1593
Epa	17.86	0.12	122	**3.56**	0.85	1779
Eva	12.67	0.13	120	5.32	3.21	1928
ca-HepTh	54.56	1.05	127	**16.10**	2.19	6298

Datos	**Icentral**			**Bidcentral**		
	μ	γ	m	μ	γ	m
ego-Facebook	28.29	11.86	188	28.97	11.48	364
ca-GrQc	9.80	1.57	123	10.29	1.50	153
Epa	6.00	1.78	120	7.54	2.11	143
Eva	**0.65**	0.14	111	1.15	0.17	135
ca-HepTh	52.44	9.51	156	48.08	8.35	216

Tabla 3.3: Resultados para los grafos reales dirigidos en ambiente incremental. Tiempo dado en segundos (media μ y desviación estándar γ) y memoria en MBytes (m).

Datos	**Brandes**			**Ibet**			**Bidcentral**		
	μ	γ	m	μ	γ	m	μ	γ	m
email-Eu-core	1.05	0.07	102	**0.39**	0.10	159	0.99	0.49	127
CollegeMsg	2.19	0.06	109	**0.61**	0.17	337	1.60	1.01	130
p2p-Gnutella08	12.52	0.38	125	**2.32**	1.47	3013	7.75	5.99	157
Wiki-Vote	22.25	1.17	175	**2.31**	3.24	4642	7.45	10.33	296
p2p-Gnutella09	19.40	0.43	135	**2.92**	1.72	4990	9.98	8.11	170
p2p-Gnutella05	25.03	0.12	143	**4.54**	2.74	6274	18.56	14.52	181
p2p-Gnutella04	42.51	1.70	153	**7.14**	3.78	8169	29.36	21.41	202
wiki-RfA	68.90	0.28	249	**6.27**	10.04	9286	22.33	35.05	468

Los resultados para grafos dirigidos reales se muestran en la Tabla 3.3. En este caso no es posible utilizar el algoritmo **Icentral**, ya que este solo puede procesar grafos no dirigidos.

Ibet es el más eficiente desde el punto de vista temporal en todos los casos. Además, **Bidcentral** es siempre más rápido que **Brandes**. La desviación estándar promedio es siempre mayor en el caso de **Bidcentral**. Esto se debe a que el tamaño del componente afectado no es constante, variando así el tiempo de ejecución. Esta situación no ocurre con el resto de los algoritmos, sobre todo en el caso de **Brandes** que tiene un desempeño relativamente constante. Como promedio, en este experimento **Bidcentral** fue 1.86 veces más rápido que **Brandes**. Respecto a la memoria utilizada, al comparar **Ibet** con **Bidcentral**, observamos como, a medida que los grafos son de mayor tamaño, la diferencia entre ambos crece sustancialmente.

Los resultados para grafos dirigidos sintéticos se muestran en la Tabla 3.4. En este caso no es posible utilizar el algoritmo **Icentral** ya que este solo puede procesar grafos no dirigidos.

Los comentarios realizados anteriormente para los grafos reales son veraces también para los grafos sintéticos. En este caso **Bidcentral** fue 1.53 veces más rápido que **Brandes** en promedio. En la Figura 3.1 y la Figura 3.2 podemos ver como varían el tiempo de ejecución y la memoria utilizada a medida que se procesan grafos sintéticos de mayor tamaño.

En este experimento con datos sintéticos realizamos además un análisis estadístico de los resultados. Los tiempos de ejecución pueden ser comparados de manera estadística utilizando la prueba de Friedman [17]. En este caso, el valor p resultó igual a 3.89×10^{-15}, por lo que se rechaza la hipótesis nula de que no existen diferencias significativas entre los tres algoritmos. Por tanto, aplicamos las pruebas post-hoc de Nemenyi y de Bonferroni-Dunn. Para la primera prueba obtuvimos la gráfica que se muestra en la Figura 3.3, donde queda reflejada la superioridad de **Bidcentral** respecto a **Brandes**. La segunda prueba, que se muestra en la Figura 3.4, mostró

Tabla 3.4: Resultados para los grafos sintéticos dirigidos en ambiente incremental. Tiempo dado en segundos (media μ y desviación estándar γ) y memoria en MBytes (m).

Datos	**Brandes**			**Ibet**			**Bidcentral**		
	μ	γ	m	μ	γ	m	μ	γ	m
RMAT-1024	0.35	0.01	92	**0.20**	0.07	108	0.56	0.19	105
RMAT-2048	0.84	0.01	103	**0.32**	0.16	176	0.86	0.53	121
RMAT-3072	1.55	0.06	111	**0.48**	0.25	261	1.34	0.94	131
RMAT-4096	2.75	0.01	117	**0.71**	0.44	470	2.17	1.78	153
RMAT-5120	4.59	0.15	122	**1.07**	0.73	634	3.42	3.08	159
RMAT-6144	5.52	0.35	130	**1.13**	0.73	768	3.84	3.78	168
RMAT-7168	8.79	0.06	140	**1.99**	1.63	1464	6.56	6.61	182
RMAT-8192	10.67	0.16	145	**1.81**	1.42	1360	5.79	6.31	188
RMAT-9216	13.24	0.07	151	**2.03**	1.61	1650	7.23	8.16	198
RMAT-10240	18.50	0.95	160	**3.05**	2.40	2473	11.86	12.70	215
RMAT-11264	22.23	0.99	166	**3.40**	2.47	2572	13.13	13.93	236
RMAT-12288	26.98	0.83	175	**4.41**	3.57	3129	17.12	17.52	245
RMAT-13312	24.21	0.18	177	**3.48**	2.92	2996	12.23	14.02	252
RMAT-14336	28.46	0.14	186	**3.99**	3.08	3669	13.14	14.53	262
RMAT-15360	34.26	0.20	194	**4.77**	4.25	4035	17.92	21.92	294
RMAT-16384	42.83	1.26	200	**6.41**	4.73	4941	25.13	26.33	346

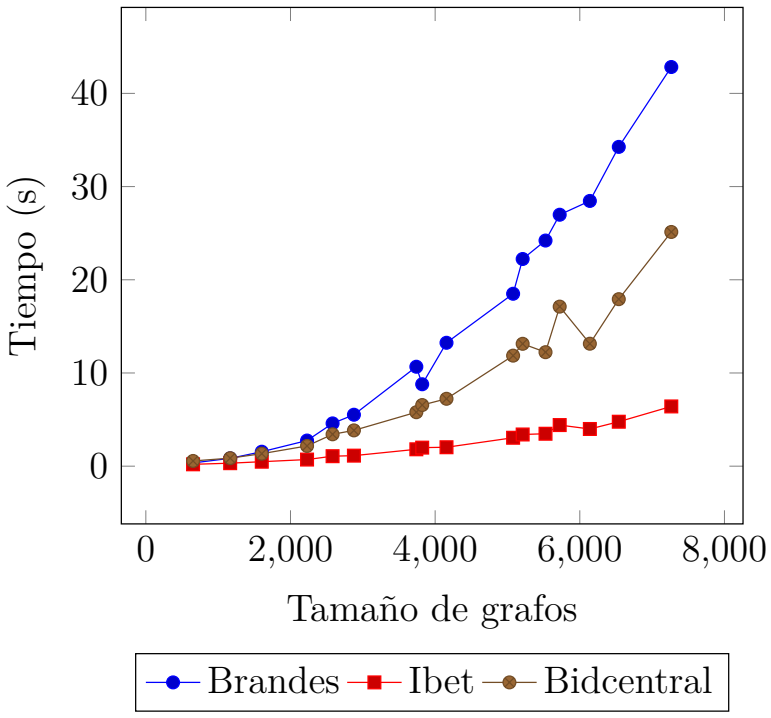

Figura 3.1: Tiempo de ejecución promedio (s) para los grafos sintéticos dirigidos en ambiente incremental

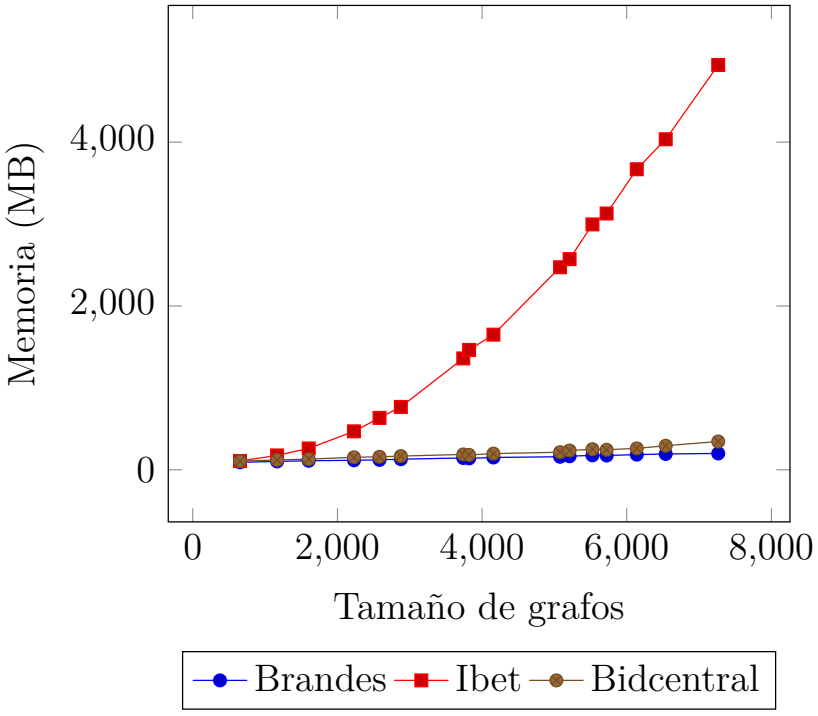

Figura 3.2: Máxima memoria utilizada (MB) para los grafos sintéticos dirigidos en ambiente incremental

resultados similares.

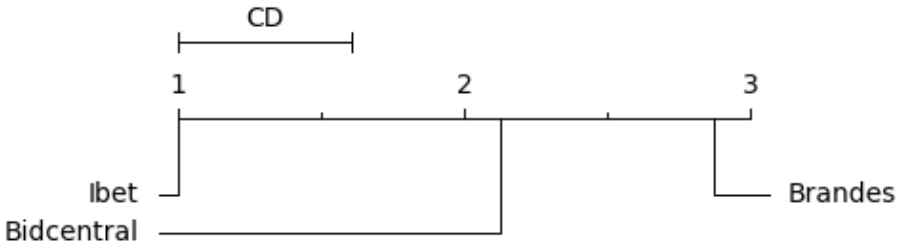

Figura 3.3: Resultados prueba de Nemenyi sobre grafos sintéticos incrementales.

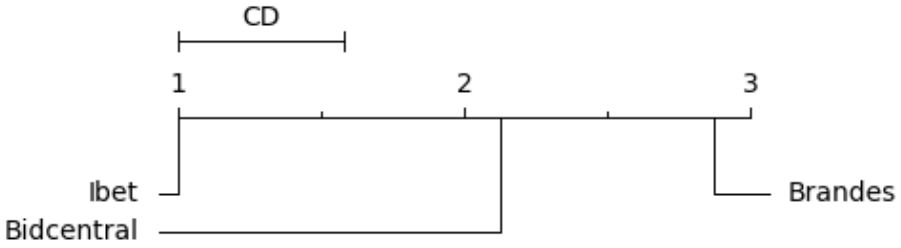

Figura 3.4: Resultados prueba de Bonferroni-Dunn sobre grafos sintéticos incrementales.

En general, teniendo en cuenta todos los conjuntos de datos, **Bidcentral** fue 1.91 veces más eficiente que **Brandes** en promedio.

Experimentos en grafos decrementales

En los últimos tres experimentos, se comparó **Bidcentral** en su faceta decremental con **Brandes** e **Icentral**. El algoritmo **Ibet** no pudo ser utilizado ya que este no permite la eliminación de aristas. Cada experimento se corresponde a un tipo de grafo particular: no dirigido real, dirigido real y dirigido sintético. Para ello, para cada grafo, se seleccionaron 100 aristas de manera aleatoria, que estuvieran contenidas previamente en el grafo, y se eliminaron de manera independiente.

Tabla 3.5: Resultados, tiempo dado en segundos (media μ y desviación estándar γ) y memoria en MBytes (m).

Datos	**Brandes**			**Icentral**			**Bidcentral**		
	μ	γ	m	μ	γ	m	μ	γ	m
ego-Facebook	21.84	0.17	152	**4.63**	9.23	187	6.63	9.27	362
ca-GrQc	9.53	0.15	109	**4.21**	3.61	123	5.36	4.13	153
Epa	9.04	0.07	110	**4.34**	2.57	120	5.49	3.07	144
Eva	6.50	0.06	109	**0.25**	0.12	114	0.52	0.12	129
ca-HepTh	50.34	0.44	126	**31.81**	17.16	155	33.01	17.45	211

Los resultados para grafos no dirigidos reales se muestran en la Tabla 3.5. De la misma manera que en el primer experimento, para utilizar el algoritmo **Bidcentral**, que trabaja con grafos dirigidos, se transformó cada arista no dirigida en un par de aristas, una para cada dirección.

Como se esperaba, **Icentral** y **Bidcentral** tuvieron una eficiencia similar, en tiempo y espacio, y fueron consistentemente más rápidos que **Brandes**. Esta mejora es altamente dependiente del tamaño del componente afectado, lo que se refleja en los resultados en Eva, donde el número de nodos en el mayor componente biconexo es relativamente pequeño. **Bidcentral** fue 3.80 veces más rápido que **Brandes** en promedio.

Los resultados para grafos dirigidos reales se muestran en la Tabla 3.6. En este caso no es posible utilizar el algoritmo **Icentral** ya que este solo puede procesar grafos no dirigidos.

Tabla 3.6: Resultados, tiempo dado en segundos (media μ y desviación estándar γ) y memoria en MBytes (m).

Datos	**Brandes**			**Bidcentral**		
	μ	γ	m	μ	γ	m
email-Eu-core	1.00	0.01	102	**0.83**	0.30	124
CollegeMsg	1.99	0.03	110	**1.78**	0.78	129
p2p-Gnutella08	11.28	0.07	125	**4.25**	4.48	156
Wiki-Vote	20.58	0.13	174	**5.33**	4.50	297
p2p-Gnutella09	19.17	0.09	135	**7.27**	7.26	172
p2p-Gnutella05	25.00	0.30	141	**8.91**	10.95	183
p2p-Gnutella04	40.91	0.20	153	**18.75**	16.82	202
wiki-RfA	69.90	0.19	252	**12.95**	13.21	468

Bidcentral tuvo un menor tiempo de ejecución en todos los casos, siendo esta vez 2.73 veces más rápido que **Brandes** en promedio.

Los resultados para grafos dirigidos sintéticos se muestran en la Tabla 3.7. Como en el experimento anterior, en este caso no es posible utilizar el algoritmo **Icentral**.

Nuevamente, los comentarios realizados anteriormente para los grafos reales son veraces para los grafos sintéticos. En este caso **Bidcentral** fue 2.07 veces más rápido que **Brandes** en promedio. En la Figura 3.5 y la Figura 3.6 podemos ver como varían el tiempo de ejecución y la memoria utilizada a medida que se procesan grafos sintéticos de mayor tamaño.

En la mayoría de los grafos dirigidos, **Bidcentral** es más eficiente que **Brandes**, y su uso de memoria es proporcional. Solo en el caso de los dos conjuntos de datos

Tabla 3.7: Resultados, tiempo dado en segundos (media μ y desviación estándar γ) y memoria en MBytes (m).

Datos	**Brandes**			**Bidcentral**		
	μ	γ	m	μ	γ	m
RMAT-1024	**0.36**	0.01	91	0.53	0.14	104
RMAT-2048	**0.85**	0.01	102	0.87	0.41	119
RMAT-3072	1.56	0.04	113	**1.45**	0.83	131
RMAT-4096	2.77	0.01	116	**1.52**	1.04	149
RMAT-5120	4.80	0.12	123	**2.56**	2.19	161
RMAT-6144	5.88	0.13	130	**3.56**	2.70	169
RMAT-7168	9.57	0.89	140	**3.80**	3.81	181
RMAT-8192	10.70	0.07	145	**4.16**	4.22	188
RMAT-9216	16.29	0.67	151	**5.60**	6.31	198
RMAT-10240	21.94	0.87	159	**6.66**	7.23	214
RMAT-11264	22.18	0.45	166	**9.16**	8.88	238
RMAT-12288	27.09	0.60	174	**11.77**	11.97	245
RMAT-13312	24.53	0.18	177	**11.17**	11.04	254
RMAT-14336	29.23	0.18	186	**13.39**	12.98	264
RMAT-15360	35.54	0.21	192	**16.75**	15.13	291
RMAT-16384	42.03	0.74	200	**16.16**	17.87	343

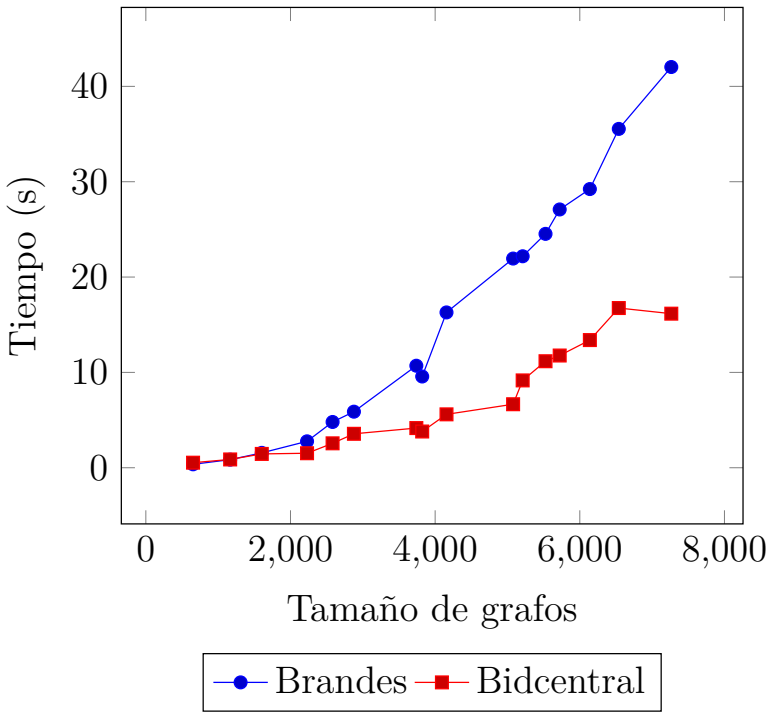

Figura 3.5: Tiempo de ejecución promedio (s) para los grafos sintéticos dirigidos en ambiente decremental

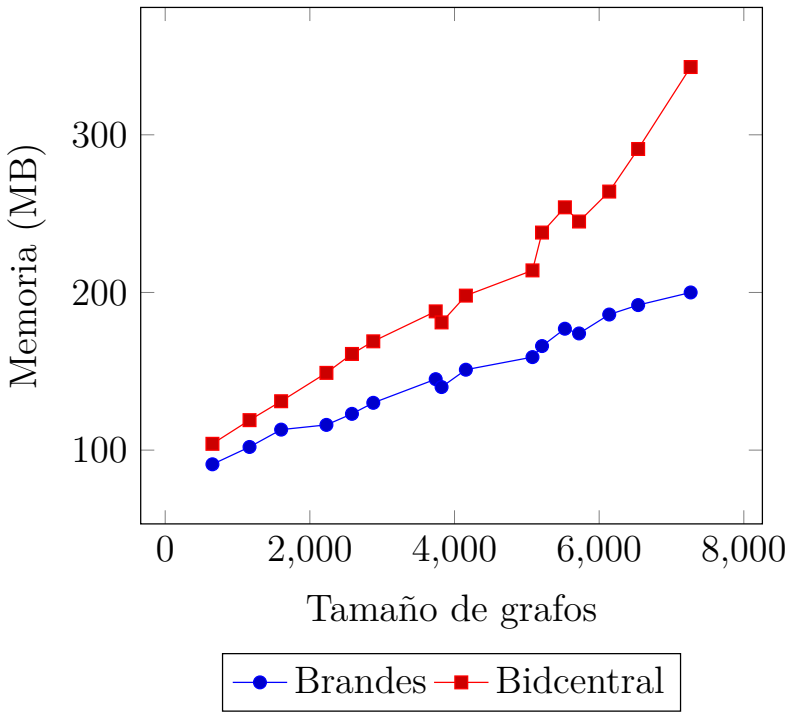

Figura 3.6: Máxima memoria utilizada (MB) para los grafos sintéticos dirigidos en ambiente decremental

aleatorios más pequeños, el desempeño de **Brandes** y nuestro algoritmo es muy similar. Esto podría estar relacionado con el hecho de que hallar los componentes biconexos agrega algunas operaciones que solo influyen de manera notable en el tiempo de ejecución cuando los grafos son muy pequeños. En los datos reales esta situación no ocurrió en la mayoría de los casos, porque en estos grafos el tamaño del mayor componente biconexo es usualmente menor.

En el último experimento (con datos sintéticos), realizamos además un análisis estadístico de los resultados. Los tiempos de ejecución pueden ser comparados de manera estadística utilizando la prueba de los rangos con signo de Wilcoxon [17]. En este caso, el valor p resultó igual a 9.35×10^{-4}, por lo que se rechaza la hipótesis nula de que no existen diferencias significativas entre los algoritmos.

En general, teniendo en cuenta todos los conjuntos de datos, **Bidcentral** fue 2.59 veces más eficiente que **Brandes** en promedio. Comparando los experimentos incrementales con los decrementales, notamos que **Bidcentral** obtiene mejores resultados respecto a **Brandes** en el segundo caso. Esto se contrapone a lo usual en la literatura, donde los algoritmos decrementales usualmente tienen mayor complejidad computacional que sus contrapartes incrementales. En el caso del algoritmo propuesto este comportamiento no es sorprendente, ya que al insertarse aristas, el tamaño de los componentes afectados puede crecer, cuestión que no ocurre así cuando se eliminan.

3.4. Conclusiones parciales

La realización de experimentos sobre datos sintéticos y reales permite corroborar la eficiencia de los algoritmos en un ambiente más cercano a la práctica, y complementan los análisis teóricos. En este capítulo se mostraron y analizaron los resultados de los experimentos realizados con algoritmos propuestos en la literatura y el objeto de esta tesis, **Bidcentral**. Para ello se utilizaron grafos extraídos de redes sociales

con distintos tamaños y orígenes, así como grafos generados aleatoriamente.

En general, **Bidcentral** fue 1.91 y 2.59 más eficiente desde el punto de vista temporal que **Brandes** en los experimentos con grafos incrementales y decrementales respectivamente. Esta mejoría probó ser significativa desde el punto de vista estadístico en el caso de los experimentos con grafos sintéticos. **Bidcentral** fue además notablemente más eficiente desde el punto de vista espacial respecto al algoritmo recientemente propuesto **Ibet**. De esta manera, queda constatado que la propuesta constituye una alternativa algorítmica para el cálculo de la intermediación en grafos dinámicos, sobre todo cuando los recursos computacionales (memoria) están acotados y se desean procesar grafos grandes.

Conclusiones

Como resultado de esta investigación se desarrolló el algoritmo **Bidcentral** para calcular la intermediación en grafos dirigidos dinámicos que permite a los analistas determinar los actores más importantes en redes sociales de gran tamaño. De esta forma se cumple el objetivo general planteado, ya que:

- Se han propuesto numerosos algoritmos para el cálculo de la intermediación en grafos extraídos de redes sociales. En el caso de los algoritmos dinámicos una característica común es que tienen una complejidad espacial cuadrática. Esto hace que no sea posible su aplicación cuando los recursos son limitados. Al respecto, se identificó que la descomposición en componentes biconexos es una técnica factible para disminuir la complejidad espacial. Esta variante ya fue realizada en **Icentral** para grafos conexos no dirigidos.

- El algoritmo **Bidcentral** utiliza la descomposición en componentes biconexos para calcular la intermediación sobre el caso más general de grafos dinámicos dirigidos no necesariamente conexos. De esta manera se mantienen la complejidad espacial lineal y la complejidad temporal dependiente del tamaño del componente biconexo más grande, características esenciales de **Icentral**.

- Los experimentos realizados sobre redes sociales de distintos tamaños y grafos generados de manera aleatoria permiten concluir que **Bidcentral** tiene mejor desempeño al procesar grafos dirigidos en cuanto a tiempo de ejecución, cuando

el espacio de almacenamiento es limitado. Por ello, este algoritmo puede ser de interés práctico para su uso en grafos dinámicos dirigidos de gran tamaño, sobre todo cuando algoritmos más eficientes no son factibles dada su complejidad espacial cuadrática.

Recomendaciones

- Conducir experimentos con una implementación distribuida y paralela del algoritmo propuesto sobre la plataforma *spark* [7].
- Incluir alguna de las heurísticas utilizadas en [5] para mejorar la eficiencia.
- Valorar la factibilidad de utilizar la descomposición en componentes fuertemente conexos, que son usualmente más pequeños que los componentes biconexos.

[7] `https://spark.apache.org/`

Referencias bibliográficas

[1] Ajith Abraham, Aboul-Ella E Hassanien, and Vaclav Snášel. *Computational Social Network Analysis: Trends, Tools and Research Advances*. Springer, 2009. Series Title: Computer Communications and Networks.

[2] Jac M Anthonisse. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, 1971.

[3] Albert-László Barabási. *Network Science*. Cambridge University Press, 2016.

[4] Alex Bavelas. A mathematical model for group structures. *Human Organization*, 7(3):16–30, 1948.

[5] Elisabetta Bergamini, Henning Meyerhenke, Mark Ortmann, and Arie Slobbe. Faster Betweenness Centrality Updates in Evolving Networks. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75, pages 1–16, 2017.

[6] Stephen P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55–71, 2005.

[7] Ulrik Brandes. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[8] Ulrik Brandes, Linton C Freeman, and Dorothea Wagner. Social networks. In *Social Networks*. 2010.

[9] Piotr Bródka, Stanisław Saganowski, and Przemysław Kazienko. Community Evolution. In *Encyclopedia of Social Network Analysis and Mining*, pages 220–232. 2014.

[10] Charu C.Aggarwal. *Social Network Data Analytics*. 2011. Publication Title: Vasa.

[11] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A Recursive Model for Graph Mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446, 2004.

[12] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.

[13] C Demetrescu and GF Italiano. Fully dynamic transitive closure: Breaking through the O (n/sup 2/) barrier. *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, (1), 2000.

[14] Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. Dynamic Graph Algorithms. In *Algorithms and Theory of Computation Handbook*, pages 9–9. Citeseer, 2010.

[15] Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004.

[16] Camil Demetrescu and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Transactions on Algorithms*, 2(4):578–601, 2006.

[17] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7:1–30, 2006.

[18] Shlomi Dolev, Yuval Elovici, Rami Puzis, and Polina Zilberman. Incremental deployment of network monitors based on Group Betweenness Centrality. *Information Processing Letters*, 109(20):1172–1176, 2009.

[19] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*, volume 81. 2010. Publication Title: Science.

[20] P Erdös and a Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[21] Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, March 1977.

[22] Linton C Freeman. *The Development of Social Network Analysis*, volume 27. 2004. Publication Title: Document Design.

[23] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully Dynamic Algorithms for Maintaining Shortest Paths Trees. *J. Algorithms*, 34(2):251–281, February 2000. Place: Duluth, MN, USA.

[24] Keshav Goel, RishiRanjan Ranjan Singh, Sudarshan Iyengar, and Sukrit. A faster algorithm to update betweenness centrality after node alteration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8305 LNCS:170–184, 2013.

[25] Oded Green, Robert McColl, David A. Bader, O. Green, R. McColl, D.A. Bader, Oded Green, Robert McColl, and David A. Bader. A Fast Algorithm for Incremental Betweenness Centrality. *ASE/IEEE International Conference on Social Computing (SocialCom)*, pages 11–20, 2012. Place: Washington, DC, USA.

[26] Matthew W. Hahn and Andrew D. Kern. Comparative genomics of centrality and essentiality in three eukaryotic protein-interaction networks. *Molecular Biology and Evolution*, 22(4):803–806, 2005.

[27] F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7):79–87, 1997.

[28] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. Fully Dynamic Betweenness Centrality Maintenance on Massive Networks. *Proceedings of the VLDB Endowment*, 9(2):48–59, 2015.

[29] Jacob Holm, Kristian De Lichtenberg, Mikkel Thorup, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

[30] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[31] Fuad Jamour, Spiros Skiadopoulos, and Panos Kalnis. Parallel Algorithm for Incremental Betweenness Centrality on Large Graphs. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):659–672, 2017.

[32] Maliackal Poulo Joy, Amy Brock, Donald E. Ingber, and Sui Huang. High-betweenness proteins in the yeast protein interaction network. *BioMed Research International*, 2005(2):96–103, 2005.

[33] Miray Kas, Matthew Wachs, Kathleen M. Carley, and L. Richard Carley. Incremental algorithm for updating betweenness centrality in dynamically growing networks. *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining - ASONAM '13*, pages 33–40, 2013.

[34] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi. Scalable online betweenness centrality in evolving graphs. *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*, 27(9):1580–1581, 2016.

[35] Valdis E Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[36] Min Joong Lee, Sunghee Choi, and Chin Wan Chung. Efficient algorithms for updating betweenness centrality in fully dynamic graphs. *Information Sciences*, 326:278–296, 2016.

[37] Mj Min-Joong Lee, Jungmin Lee, Jy Jaimie Yejean Park, Ryan Hyun Choi, and Chin-Wan Chung. QUBE: A Quick algorithm for Updating Betweenness centrality. *WWW 2012 - Session: Community Detection in Social Networks*, pages 351–360, 2012. Place: New York, NY, USA.

[38] Loet Leydesdorff. Betweenness centrality as an indicator of the interdisciplinarity of scientific journals. *Journal of the American Society for Information Science and Technology*, 58(9):1303–1319, 2007.

[39] J. L. Moreno. Foundations of Sociometry: An Introduction. *Sociometry*, 1941.

[40] Shivaram Narayanan. *The Betweenness Centrality of Biological Networks*. PhD thesis, Virginia Polytechnic Institute and State University, 2005. Publication Title: Distribution.

[41] Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran. Betweenness Centrality - Incremental and Faster. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, {MFCS} 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part {II}*, pages 577–588, 2014.

[42] Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran. Decremental all-pairs ALL shortest paths and betweenness centrality. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8889:766–778, 2014.

[43] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010. Publication Title: Networks: An Introduction.

[44] M. Pontecorvi and V. Ramachandran. Fully dynamic betweenness centrality. In *International Symposium on Algorithms and Computation*, 2015.

[45] Matteo Pontecorvi and Vijaya Ramachandran. Fully Dynamic All Pairs All Shortest Paths. pages 1–18, 2014.

[46] Rami Puzis, Yaniv Altshuler, Yuval Elovici, Shlomo Bekhor, Yoram Shiftan, and Alex (Sandy) Pentland. Augmented Betweenness Centrality for Environmentally Aware Traffic Monitoring in Transportation Networks. *Journal of Intelligent Transportation Systems*, 17(1):91–105, 2013.

[47] Rami Puzis, Polina Zilberman, Yuval Elovici, Shlomi Dolev, and Ulrik Brandes. Heuristics for speeding up betweenness centrality computation. *Proceedings - 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust and 2012 ASE/IEEE International Conference on Social Computing, SocialCom/PASSAT 2012*, pages 302–311, 2012. Place: Washington, DC, USA.

[48] G Ramalingam and Thomas Reps. On the computational complexity of dynamic graph problems. *Theoretical Computer Science*, 158(1–2):233–277, 1996.

[49] Matteo Riondato and Eli Upfal. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, pages 1–27, 2018.

[50] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica (New York)*, 61(2):389–401, 2011.

[51] Edgar Solomonik, Maciej Besta, Flavio Vella, and Torsten Hoefler. Scaling Betweenness Centrality Using Communication-efficient Sparse Matrix Multiplication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 47:1–47:14, New York, NY, USA, 2017. ACM.

[52] Myra Spiliopoulou. Evolution in social networks: A survey. In *Social Network Data Analytics*, pages 149–175. 2011.

[53] Jimeng Sun and Jie Tang. Models and Algorithms for Social Influence Analysis. In *Proceedings of the Sixth ACM International Conference on Web Search and*

Data Mining, pages 775–776, New York, NY, USA, 2013. ACM. Series Title: WSDM '13.

[54] Ali Tizghadam and Alberto Leon-Garcia. Betweenness centrality and resistance distance in communication networks. *IEEE Network*, 24(6):10–16, 2010.

[55] Alok Tripathy and Oded Green. Scaling Betweenness Centrality in Dynamic Graphs. In *2018 IEEE High Performance Extreme Computing Conference (HPEC)*, 2018.

[56] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994. Series Title: Structural Analysis in the Social Sciences.

[57] Duncan J. Watts and Steven H. Strogatz. Collective dinamycs of 'small world' networks. *nature*, 393(6684):144–164, 1998.

[58] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018.

[59] Jeongah Yoon, Anselm Blumer, and Kyongbum Lee. An algorithm for modularity analysis of directed and weighted biological networks based on edge-betweenness centrality. *Bioinformatics*, 22(24):3106–3108, 2006.

[60] Keyou You, Roberto Tempo, and Li Qiu. Distributed Algorithms for Computation of Centrality Measures in Complex Networks. *IEEE Transactions on Automatic Control*, 62(5):2080–2094, 2017.

[61] Haiyuan Yu, Philip M. Kim, Emmett Sprecher, Valery Trifonov, and Mark Gerstein. The importance of bottlenecks in protein networks: Correlation with gene essentiality and expression dynamics. *PLoS Computational Biology*, 3(4):713–720, 2007.