

# Tabla de contenido

## Prefacio

[Que hay en este libro](#)

[Las convenciones usadas en este libro](#)

[Usando ejemplos de código](#)

[Aprendizaje en línea de O'Reilly](#)

[Cómo contactarnos](#)

[Expresiones de gratitud](#)

## Prefacio

### 1. Introducción

[¿Qué son los gráficos?](#)

[¿Qué son los análisis gráficos y los algoritmos?](#)

[Procesamiento de grafos, bases de datos, consultas y algoritmos](#)

[OLTP y OLAP](#)

[¿Por qué deberíamos preocuparnos por los algoritmos de grafos?](#)

[Casos de uso de Graph Analytics](#)

[Conclusión](#)

### 2. Teoría de la gráfica y conceptos

[Terminología](#)

[Tipos de grafos y estructuras](#)

[Estructuras aleatorias, de mundo pequeño, sin escala](#)

[Sabores de grafos](#)

[Gráficos conectados y desconectados](#)

[Gráficos no ponderados versus gráficos ponderados](#)

[Gráficos no direccionalizados versus gráficos dirigidos](#)

[Gráficos acíclicos versus gráficos cíclicos](#)

[Gráficos escasos frente a gráficos densos](#)

[Gráficos monopartitos, bipartitos y k-partitas](#)

[Tipos de algoritmos de grafos](#)

[Pathfinding](#)

[Centralidad](#)

[Detección de la comunidad](#)

[Resumen](#)

### 3. Plataformas gráficas y procesamiento.

[Plataforma gráfica y consideraciones de procesamiento](#)

[Consideraciones de la plataforma](#)

[Consideraciones de procesamiento](#)

[Plataformas representativas](#)

[Seleccionando nuestra plataforma](#)

[Chispa de apache](#)

[Plataforma gráfica Neo4j](#)

[Resumen](#)

### 4. Algoritmos de búsqueda de rutas y gráficos.

[Datos de ejemplo: el gráfico de transporte](#)

[Importando los datos en Apache Spark](#)

[Importando los datos en Neo4j](#)

[Amplia primera búsqueda](#)

[Amplia primera búsqueda con Apache Spark](#)[Primera búsqueda de profundidad](#)[Trayectoria más corta](#)[¿Cuándo debo usar el camino más corto?](#)[El camino más corto con Neo4j](#)[Ruta más corta \(ponderada\) con Neo4j](#)[Ruta más corta \(ponderada\) con Apache Spark](#)[Variación del camino más corto: A \\*](#)[Variación del camino más corto: los k-caminos más cortos de Yen](#)[El camino más corto de todos los pares](#)[Una mirada más cercana a todos los pares camino más corto](#)[¿Cuándo debo usar el camino más corto de todos los pares?](#)[El camino más corto de todos los pares con Apache Spark](#)[El camino más corto de todos los pares con Neo4j](#)[La ruta más corta de una sola fuente](#)[¿Cuándo debo usar la ruta más corta de una sola fuente?](#)[La ruta más corta de una sola fuente con Apache Spark](#)[La ruta más corta de una sola fuente con Neo4j](#)[Árbol de expansión mínima](#)[¿Cuándo debo usar el árbol de expansión mínima?](#)[Árbol de expansión mínimo con Neo4j](#)[Caminata aleatoria](#)[¿Cuándo debo usar la caminata aleatoria?](#)[Paseo aleatorio con Neo4j](#)[Resumen](#)[5. Algoritmos de centralidad.](#)[Ejemplo de datos de gráfico: El gráfico social](#)[Importando los datos en Apache Spark](#)[Importando los datos en Neo4j](#)[Grado de centralidad](#)[Alcanzar](#)[¿Cuándo debo usar la centralidad del grado?](#)[Grado de Centralidad con Apache Spark](#)[Proximidad centralidad](#)[¿Cuándo debo usar la centralidad de proximidad?](#)[Centralidad de proximidad con chispa de apache](#)[Centralidad de proximidad con Neo4j](#)[Variación de la proximidad de la centralidad: Wasserman y Fausto](#)[Variación de la proximidad de la centralidad: Centralidad armónica](#)[Centralidad de intermedición](#)[¿Cuándo debo usar la centralidad de intermedición?](#)[Centralidad de intermedición con Neo4j](#)[Variación de centralidad de la unidad: marcas aleatorias-aproximadas](#)[Rango de página](#)[Influencia](#)[La fórmula de PageRank](#)[Iteración, surfistas aleatorios y sumideros de rango](#)[¿Cuándo debo usar PageRank?](#)

[PageRank con Apache Spark](#)[PageRank con Neo4j](#)[Variación del PageRank: PageRank personalizado](#)[Resumen](#)

## 6. Algoritmos de detección de la comunidad.

[Ejemplo de datos de gráfico: el gráfico de dependencia de software](#)[Importando los datos en Apache Spark](#)[Importando los datos en Neo4j](#)[Conteo de triángulos y coeficiente de agrupamiento](#)[Coeficiente de agrupamiento local](#)[Coeficiente de agrupamiento global](#)[¿Cuándo debo usar el recuento de triángulos y el coeficiente de agrupamiento?](#)[Recuento de triángulos con chispa de apache](#)[Triángulos con Neo4j](#)[Coeficiente de agrupamiento local con Neo4j](#)[Componentes fuertemente conectados](#)[¿Cuándo debo usar componentes fuertemente conectados?](#)[Componentes fuertemente conectados con Apache Spark](#)[Componentes fuertemente conectados con Neo4j](#)[Componentes conectados](#)[¿Cuándo debo usar los componentes conectados?](#)[Componentes conectados con Apache Spark](#)[Componentes conectados con Neo4j](#)[Propagación de etiquetas](#)[Aprendizaje semi-supervisado y etiquetas de semillas](#)[¿Cuándo debo usar la propagación de etiquetas?](#)[Propagación de etiquetas con Apache Spark](#)[Propagación de etiquetas con Neo4j](#)[Modularidad de Lovaina](#)[¿Cuándo debo usar Louvain?](#)[Louvain con Neo4j](#)[Validando Comunidades](#)[Resumen](#)

## 7. Graficar algoritmos en la práctica.

[Analizando datos de Yelp con Neo4j](#)[Red social de Yelp](#)[Importación de datos](#)[Modelo gráfico](#)[Una descripción rápida de los datos de Yelp](#)[Aplicación de planificación de viaje](#)[Consultoría de viajes de negocios](#)[Encontrar categorías similares](#)[Analizando los datos de vuelo de las aerolíneas con Apache Spark](#)[Análisis exploratorio](#)[Aeropuertos populares](#)[Retrasos de ORD](#)[Mal día en la OFS](#)[Aeropuertos interconectados por aerolínea](#)

[Resumen](#)

## [8. Uso de algoritmos gráficos para mejorar el aprendizaje automático](#)

[Aprendizaje automático y la importancia del contexto](#)[Gráficos, contexto y precisión](#)[Extracción y selección de características conectadas](#)[Características de Graphy](#)[Características del algoritmo gráfico](#)[Gráficos y aprendizaje automático en la práctica: Predicción de enlaces](#)[Herramientas y datos](#)[Importando los datos en Neo4j](#)[El gráfico de coautoría](#)[Creación de conjuntos de datos de prueba y entrenamiento equilibrados](#)[Cómo predecimos los enlaces que faltan](#)[Creación de una tubería de aprendizaje automático](#)[Predicción de enlaces: características básicas del gráfico](#)[Predicción de enlaces: triángulos y el coeficiente de agrupación](#)[Predicción de enlaces: Detección comunitaria](#)[Resumen](#)[Envolviendo las cosas](#)

## [A. Información y recursos adicionales](#)

[Otros algoritmos](#)[Neo4j importación de datos a granel y Yelp](#)[APOC y otras herramientas Neo4j](#)[Encontrar conjuntos de datos](#)[Asistencia con las plataformas Apache Spark y Neo4j.](#)[Formación](#)[Índice](#)

# Neo4j

(Data Scientists) LOVE (Neo4j)



POLYGRAPH. PRACTICAL. INTELLIGENT

Neo4j is the platform for *connected data*,  
with native graph analytics, storage and  
processing. Graph algorithms in Neo4j reveal  
hidden patterns and enforce machine  
learning predictions.

Try Neo4j

neo4j

# Algoritmos de grafos

Ejemplos prácticos en  
Apache Spark y Neo4j

**Mark Needham y Amy E. Hodler**

## Algoritmos de grafos

por Mark Needham y Amy E. Hodler

Copyright © 2019 Amy Hodler y Mark Needham. Todos los derechos reservados.

Impreso en los Estados Unidos de América.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Los libros de O'Reilly se pueden comprar para uso educativo, comercial o promocional de ventas. Las ediciones en línea también están disponibles para la mayoría de los títulos (<http://oreilly.com>). Para obtener más información, póngase en contacto con nuestro departamento de ventas corporativo / institucional: 800-998-9938 o [corporate@oreilly.com](mailto:corporate@oreilly.com).

Editor de adquisiciones: Jonathan Hassell

Editor: Jeff Bleiel

Editor de producción: Deborah Baker

Editor de la copia: Tracy Brown

Corrector de pruebas: Rachel Head

Indexador: Judy McConville

Diseñador de interiores: David Futato

Diseñador de portada: Karen Montgomery

Ilustrador: Rebecca Demarest

Mayo 2019: Primera Edición

### Historial de revisiones para la primera edición

- 2019-04-15: Primer lanzamiento

Consulte <http://oreilly.com/catalog/errata.csp?isbn=9781492047681> para obtener detalles de la versión.

El logotipo de O'Reilly es una marca comercial registrada de O'Reilly Media, Inc. Graph Algorithms , la imagen de portada de una araña de jardín europea, y la imagen comercial relacionada son marcas comerciales de O'Reilly Media, Inc.

Si bien el editor y los autores han realizado esfuerzos de buena fe para garantizar que la información y las instrucciones contenidas en este trabajo sean precisas, el editor y los autores renuncian a toda responsabilidad por errores u omisiones, incluida, entre otras, la responsabilidad por los daños resultantes del uso de o Confianza en este trabajo. El uso de la información y las instrucciones contenidas en este trabajo es bajo su propio riesgo. Si cualquier muestra de código u otra tecnología que contenga o describa este trabajo está sujeta a licencias de código abierto o los derechos de propiedad intelectual de terceros, es su responsabilidad asegurarse de que su uso cumpla con dichas licencias y / o derechos.

Este trabajo es parte de una colaboración entre O'Reilly y Neo4j. Vea nuestra [declaración de independencia editorial](#).

978-1-492-05781-9

[LSI]

# Prefacio

---

El mundo está impulsado por conexiones, desde los sistemas financieros y de comunicación hasta los procesos sociales y biológicos. Revelar el significado detrás de estas conexiones impulsa avances en las industrias en áreas como la identificación de timbres de fraude y la optimización de las recomendaciones para evaluar la fuerza de un grupo y predecir fallas en cascada.

A medida que la conexión continúa acelerándose, no es sorprendente que el interés en los algoritmos de grafos haya explotado porque se basan en matemáticas desarrolladas explícitamente para obtener información de las relaciones entre los datos. El análisis gráfico puede descubrir el funcionamiento de sistemas y redes intrincados a escalas masivas, para cualquier organización.

Nos apasiona la utilidad y la importancia de la analítica gráfica, así como la alegría de descubrir el funcionamiento interno de escenarios complejos. Hasta hace poco, la adopción del análisis gráfico requería una experiencia y una determinación significativas, porque las herramientas y las integraciones eran difíciles y pocos sabían cómo aplicar algoritmos gráficos a sus dilemas. Nuestro objetivo es ayudar a cambiar esto. Escribimos este libro para ayudar a las organizaciones a aprovechar mejor el análisis gráfico para que puedan hacer nuevos descubrimientos y desarrollar soluciones inteligentes más rápido.

## Que hay en este libro

Este libro es una guía práctica para comenzar con algoritmos de gráficos para desarrolladores y científicos de datos que tienen experiencia en el uso de Apache Spark™ o Neo4j. Aunque nuestros ejemplos de algoritmos utilizan las plataformas Spark y Neo4j, este libro también será útil para comprender conceptos de gráficos más generales, independientemente de su elección de tecnologías de gráficos.

Los primeros dos capítulos proporcionan una introducción a la analítica gráfica, algoritmos y teoría. El tercer capítulo cubre brevemente las plataformas que se utilizan en este libro antes de sumergirnos en tres capítulos que se centran en los algoritmos de gráficos clásicos: búsqueda de rutas, centralidad y detección de comunidades. Concluimos el libro con dos capítulos que muestran cómo se utilizan los algoritmos de grafos dentro de los flujos de trabajo: uno para el análisis general y otro para el aprendizaje automático.

Al comienzo de cada categoría de algoritmos, hay una tabla de referencia para ayudarlo a saltar rápidamente al algoritmo relevante. Para cada algoritmo, encontrarás:

- Una explicación de lo que hace el algoritmo.
- Casos de uso para el algoritmo y referencias a donde puede obtener más información.
- Código de ejemplo que proporciona formas concretas de usar el algoritmo en Spark, Neo4j o ambos

## Las convenciones usadas en este libro

Las siguientes convenciones tipográficas se utilizan en este libro:

**Itálico**

Indica nuevos términos, direcciones URL, direcciones de correo electrónico, nombres de archivos y extensiones de archivos.

**Ancho constante**

Se usa para listas de programas, así como dentro de los párrafos para referirse a elementos del programa como nombres de funciones o variables, bases de datos, tipos de datos, variables de entorno, declaraciones y palabras clave.

**Ancho constante negrita**

Muestra comandos u otro texto que debe ser escrito literalmente por el usuario.

**Ancho constante itálica**

Muestra el texto que debe reemplazarse con valores proporcionados por el usuario o por valores determinados por contexto.

**PROPINA**

Este elemento significa una sugerencia o sugerencia.

**NOTA**

Este elemento significa una nota general.

**ADVERTENCIA**

Este elemento indica una advertencia o precaución.

## Usando ejemplos de código

El material complementario (ejemplos de código, ejercicios, etc.) está disponible para descargar en <https://bit.ly/2FPgGVV> .

Este libro está aquí para ayudarte a hacer tu trabajo. En general, si se ofrece un código de ejemplo con este libro, puede usarlo en sus programas y documentación. No necesita ponerse en contacto con nosotros para obtener permiso a menos que esté reproduciendo una parte significativa del código. Por ejemplo, escribir un programa que usa varios trozos de código de este libro no requiere permiso. Vender o distribuir un CD-ROM de ejemplos de los libros de O'Reilly requiere permiso. Responder una pregunta citando este libro y citando el código de ejemplo no requiere permiso. La incorporación de una cantidad significativa de código de ejemplo de este libro en la documentación de su producto requiere permiso.

Apreciamos, pero no exigimos, la atribución. Una atribución generalmente incluye el título, el autor, el editor y el ISBN. Por ejemplo: " Graph Algorithms de Amy E. Hodler y Mark Needham (O'Reilly). Copyright 2019 Amy E. Hodler y Mark Needham, 978-1-492-05781-9 ".

Si cree que el uso de los ejemplos de código queda fuera del uso legítimo o del permiso que se indica anteriormente, no dude en contactarnos en [permissions@oreilly.com](mailto:permissions@oreilly.com) .

## Aprendizaje en línea de O'Reilly

**NOTA**

Durante casi 40 años, [O'Reilly](#) ha brindado capacitación en tecnología y negocios, conocimiento y conocimiento para ayudar a las empresas a tener éxito.

Nuestra red única de expertos e innovadores comparte su conocimiento y experiencia a través de libros, artículos, conferencias y nuestra plataforma de aprendizaje en línea. La plataforma de aprendizaje en línea de O'Reilly le brinda acceso a cursos de capacitación en vivo, rutas de aprendizaje en profundidad, entornos de codificación interactivos y una amplia colección de textos y videos de O'Reilly y más de 200 editores. Para obtener más información, visite <http://oreilly.com> .

## Cómo contactarnos

Por favor, dirija los comentarios y preguntas sobre este libro a la editorial:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (en los Estados Unidos o Canadá)

707-829-0515 (internacional o local)

707-829-0104 (fax)

Tenemos una página web para este libro, donde enumeramos erratas, ejemplos y cualquier información adicional. Puede acceder a esta página en <http://bit.ly/graph-algorithms>.

Para comentar o hacer preguntas técnicas sobre este libro, envíe un correo electrónico a [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com).

Para obtener más información sobre nuestros libros, cursos, conferencias y noticias, visite nuestro sitio web en <http://www.oreilly.com>.

Encuéntrenos en Facebook: <http://facebook.com/oreilly>

Síguenos en Twitter: <http://twitter.com/oreillymedia>

Míranos en YouTube: <http://www.youtube.com/oreillymedia>

## Expresiones de gratitud

Hemos disfrutado mucho armando el material para este libro y agradecemos a todos los que asistieron. Nos gustaría agradecer especialmente a Michael Hunger por su orientación, a Jim Webber por sus valiosas ediciones, ya Tomaz Bratanic por su investigación. Finalmente, apreciamos enormemente que Yelp nos permita usar su rico conjunto de datos para obtener ejemplos poderosos.

# Prefacio

---

¿Qué tienen en común las siguientes cosas: análisis de atribución de mercadeo, análisis contra el lavado de dinero (AML), modelado de la trayectoria del cliente, análisis de factor causal de incidentes de seguridad, descubrimiento basado en la literatura, detección de redes de fraude, análisis de nodos de búsqueda en Internet, creación de aplicaciones de mapas? , análisis de agrupamiento de enfermedades y análisis del rendimiento de una obra de William Shakespeare. Como habrás adivinado, lo que estos tienen en común es el uso de gráficos, lo que demuestra que Shakespeare tenía razón cuando declaró: "Todo el mundo es un gráfico!"

Bien, el Bardo de Avon no escribió realmente el gráfico en esa oración, escribió la etapa . Sin embargo, observe que los ejemplos enumerados anteriormente involucran a todas las entidades y las relaciones entre ellas, incluidas las relaciones directas e indirectas (transitivas). Las entidades son los nodos del gráfico: pueden ser personas, eventos, objetos, conceptos o lugares. Las relaciones entre los nodos son los bordes en el gráfico. Por lo tanto, ¿no es la esencia misma de una representación de Shakespeare la representación activa de las entidades (los nodos) y sus relaciones (los bordes)? En consecuencia, tal vez Shakespeare podría haber escrito un gráfico en su famosa declaración.

Lo que hace que los algoritmos de gráficos y las bases de datos de gráficos sean tan interesantes y potentes no es la simple relación entre dos entidades, con A relacionada con B. Después de todo, el modelo relacional estándar de las bases de datos creaba instancias de este tipo de relaciones en sus cimientos hace décadas, en el Diagrama de relaciones de entidades (ERD). Lo que hace que los gráficos sean tan importantes son las relaciones direccionales y las relaciones transitivas. En las relaciones direccionales, A puede causar B, pero no lo contrario. En las relaciones transitivas, A puede relacionarse directamente con B y B puede relacionarse directamente con C, mientras que A no está directamente relacionada con C, de modo que, en consecuencia, A está relacionada de manera transitiva con C.

Con estas relaciones de transitividad, particularmente cuando son numerosas y diversas, con muchos posibles patrones de relación / red y grados de separación entre las entidades, el modelo gráfico descubre relaciones entre entidades que de otra manera podrían parecer desconectadas o no relacionadas, y no son detectadas por una base de datos relacional . Por lo tanto, el modelo gráfico se puede aplicar de manera productiva y eficaz en muchos casos de uso de análisis de red.

Considere este caso de uso de atribución de marketing: la persona A ve la campaña de marketing; la persona A habla de ello en las redes sociales; la persona B está conectada a la persona A y ve el comentario; y, posteriormente, la persona B compra el producto. Desde la perspectiva del gerente de campaña de marketing, el modelo relacional estándar no identifica la atribución, ya que B no vio la campaña y A no respondió a la campaña. La campaña parece un fracaso, pero su éxito real (y ROI positivo) se descubre mediante el algoritmo de análisis gráfico a través de la relación transitiva entre la campaña de marketing y la compra del cliente final, a través de un intermediario (entidad en el centro).

Luego, considere un caso de análisis contra el lavado de dinero (AML): las personas A y C son sospechosas de tráfico ilícito. Cualquier interacción entre los dos (por ejemplo, una transacción financiera en una base de datos financiera) será marcada por las autoridades y examinada en detalle. Sin embargo, si A y C nunca realizan transacciones comerciales en conjunto, sino que realizan transacciones financieras a través de una autoridad financiera B segura, respetada y sin trabas, ¿qué podría recoger la transacción? El algoritmo de análisis gráfico! El motor gráfico descubrirá la relación transitiva entre A y C a través del intermediario B.

En las búsquedas en Internet, los principales motores de búsqueda utilizan un algoritmo de red con hipervínculos (basado en gráficos) para encontrar el nodo central autorizado en todo el Internet para cualquier conjunto dado de palabras de búsqueda. La direccionalidad del borde es vital en este caso, ya que el nodo autoritario en la red es el que apuntan muchos otros nodos.

Con el descubrimiento basado en la literatura (LBD): una aplicación de red de conocimiento (basada en gráficos) que permite descubrimientos significativos a través de la base de conocimiento de miles (o incluso

millones) de artículos de revistas de investigación: el “conocimiento oculto” se descubre solo a través de la conexión entre la investigación publicada. resultados que pueden tener muchos grados de separación (relaciones transitivas) entre ellos. La LBD se está aplicando a estudios de investigación sobre el cáncer, donde la base de conocimiento médico semántico masivo de síntomas, diagnósticos, tratamientos, interacciones de medicamentos, marcadores genéticos, resultados a corto plazo y consecuencias a largo plazo podría ser "esconder" curas o tratamientos beneficiosos previamente desconocidos Para los casos más impenetrables. El conocimiento ya podría estar en la red, pero necesitamos conectar los puntos para encontrarlo.

Se pueden dar descripciones similares del poder de la gráfica para los otros casos de uso enumerados anteriormente, todos los ejemplos de análisis de red a través de algoritmos gráficos. Cada caso involucra profundamente a las entidades (personas, objetos, eventos, acciones, conceptos y lugares) y sus relaciones (puntos de contacto, asociaciones causales y simples).

Al considerar el poder de los gráficos, debemos tener en cuenta que tal vez el nodo más poderoso en un modelo gráfico para casos de uso en el mundo real podría ser el "contexto". El contexto puede incluir el tiempo, la ubicación, los eventos relacionados, las entidades cercanas y más. La incorporación de contexto en el gráfico (como nodos y como bordes) puede producir un impresionante análisis predictivo y capacidades de análisis prescriptivo.

Los algoritmos de grafos de Mark Needham y Amy Hodler tienen como objetivo ampliar nuestros conocimientos y capacidades en torno a estos importantes tipos de análisis de gráficos, incluidos algoritmos, conceptos y aplicaciones prácticas de aprendizaje automático de los algoritmos. Desde los conceptos básicos hasta los algoritmos fundamentales, las plataformas de procesamiento y los casos de uso práctico, los autores han compilado una guía instructiva e ilustrativa del maravilloso mundo de los gráficos.

Kirk Borne, PhD

Científico principal de datos y asesor ejecutivo

Booz Allen Hamilton

Marzo 2019

# Capítulo 1 Introducción

Los gráficos son uno de los temas unificadores de la informática, una representación abstracta que describe la organización de los sistemas de transporte, las interacciones humanas y las redes de telecomunicaciones. Que se puedan modelar tantas estructuras diferentes utilizando un solo formalismo es una fuente de gran poder para el programador educado.

El Manual de Diseño de Algoritmos , por Steven S. Skiena (Springer), Profesor Distinguido de Ciencias de la Computación en la Universidad de Stony Brook

Los desafíos de datos más apremiantes de hoy se centran en las relaciones, no solo en la tabulación de datos discretos. Las tecnologías gráficas y los análisis brindan herramientas poderosas para los datos conectados que se utilizan en investigación, iniciativas sociales y soluciones empresariales, tales como:

- Modelado de entornos dinámicos desde mercados financieros a servicios de TI.
- Previsión de la propagación de las epidemias, así como los atrasos e interrupciones del servicio.
- Encontrar características predictivas para el aprendizaje automático para combatir los delitos financieros.
- Descubriendo patrones para experiencias personalizadas y recomendaciones.

A medida que los datos se interconectan cada vez más y los sistemas son cada vez más sofisticados, es esencial hacer uso de las relaciones ricas y en evolución dentro de nuestros datos.

Este capítulo proporciona una introducción al análisis de gráficos y algoritmos de gráficos. Comenzaremos con un breve repaso sobre el origen de los gráficos antes de introducir algoritmos de gráficos y explicar la diferencia entre las bases de datos de gráficos y el procesamiento de gráficos. Exploraremos la naturaleza de los datos modernos en sí, y cómo la información contenida en las conexiones es mucho más sofisticada de lo que podemos descubrir con métodos estadísticos básicos. El capítulo concluirá con un vistazo a los casos de uso donde se pueden emplear algoritmos de grafos.

## ¿Qué son los gráficos?

Los gráficos tienen una historia que se remonta a 1736, cuando Leonhard Euler resolvió el problema de los "Siete puentes de Königsberg". El problema preguntó si era posible visitar las cuatro áreas de una ciudad conectadas por siete puentes, mientras que solo cruzaban cada puente una vez. No fue

Con la idea de que solo las conexiones en sí eran relevantes, Euler sentó las bases de la teoría de grafos y sus matemáticas. [La figura 1-1](#) muestra la progresión de Euler con uno de sus bocetos originales, del artículo ["Solutio problematis ad geometriam situs pertinentis"](#).

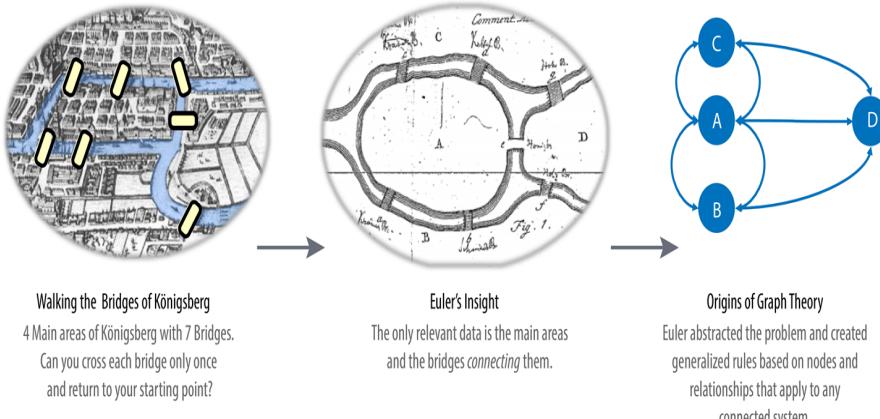


Figura 1-1. Los orígenes de la teoría de grafos. La ciudad de [Königsberg](#) incluía dos grandes islas conectadas entre sí y las dos partes continentales de la ciudad por siete puentes. El rompecabezas consistía en crear un paseo por la ciudad, cruzando cada puente una y solo una vez.

Si bien los gráficos se originaron en las matemáticas, también son una forma pragmática y de alta fidelidad de modelar y analizar datos. Los objetos que forman un gráfico se denominan nodos o vértices y los enlaces entre ellos se conocen como relaciones, enlaces o aristas. Usamos los términos nodos y relaciones en este libro: puedes pensar en los nodos como los sustantivos en las oraciones, y Las relaciones como verbos dan contexto a los nodos. Para evitar confusiones, las gráficas de las que hablamos en este libro no tienen nada que ver con las ecuaciones gráficas o los gráficos, como se muestra en la [Figura 1-2](#).

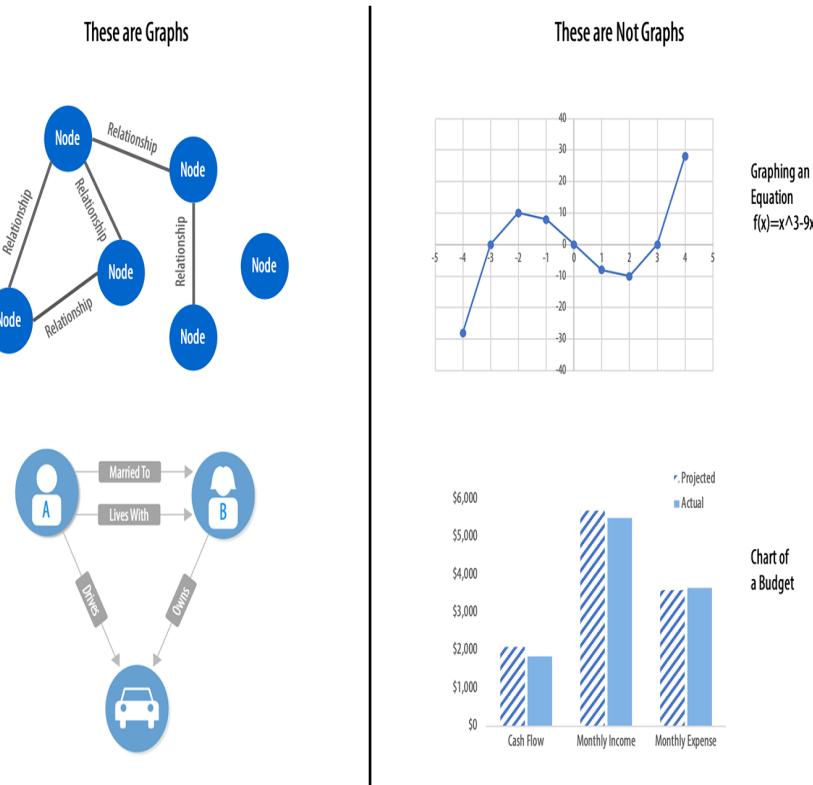


Figura 1-2. Un gráfico es una representación de una red, a menudo ilustrada con círculos para representar entidades a las que llamamos nodos, y líneas para representar relaciones.

Al observar el gráfico de personas en la [Figura 1-2](#), podemos construir fácilmente varias oraciones que lo describan. Por ejemplo, la persona A vive con la persona B que posee un automóvil, y la persona A maneja un automóvil que la persona B posee. Este enfoque de modelado es convincente porque se asigna fácilmente al mundo real y es muy "compatible con la pizarra". Esto ayuda a alinear el modelado y análisis de datos.

Pero modelar gráficas es solo la mitad de la historia. También podríamos querer procesarlos para revelar información que no es inmediatamente obvia. Este es el dominio de los algoritmos gráficos.

## ¿Qué son los análisis gráficos y los algoritmos?

Grafico Los algoritmos son un subconjunto de herramientas para el análisis gráfico. El análisis gráfico es algo que hacemos, es el uso de cualquier enfoque basado en gráficos para analizar datos conectados. Hay varios métodos que podríamos usar: podríamos consultar los datos del gráfico, usar estadísticas básicas, explorar visualmente los gráficos o incorporar gráficos en nuestras tareas de aprendizaje automático. Las consultas basadas en patrones de gráficos se utilizan a menudo para el análisis de datos locales, mientras que los algoritmos computacionales de gráficos generalmente se refieren a un análisis más global e iterativo. Aunque hay una superposición en la forma en que se pueden emplear estos tipos de análisis, usamos el término algoritmos de grafos para referirnos a los últimos, más analíticas computacionales y usos de la ciencia de datos.

Los algoritmos de grafos proporcionan uno de los enfoques más potentes para analizar datos conectados porque sus cálculos matemáticos se construyen específicamente para operar en relaciones. Describen los pasos a seguir para procesar un gráfico para descubrir sus cualidades generales o cantidades específicas. Basados en las matemáticas de la teoría de grafos, los algoritmos de grafos utilizan las relaciones entre nodos

para inferir la organización y la dinámica de sistemas complejos. Los científicos de redes utilizan estos algoritmos para descubrir información oculta, probar hipótesis y hacer predicciones sobre el comportamiento.

## RED DE CIENCIAS

La ciencia de redes es un campo académico fuertemente arraigado en la teoría de grafos que se ocupa de los modelos matemáticos de las relaciones entre objetos. Los científicos de redes se basan en algoritmos de gráficos y sistemas de administración de bases de datos debido al tamaño, la conexión y la complejidad de sus datos.

Hay muchos recursos fantásticos para la complejidad y la ciencia de la red. Aquí hay algunas referencias para que usted explore.

- [Network Science](#), de Albert-László Barabási, es un libro electrónico introductorio.
- [Complexity Explorer](#) ofrece cursos en línea.
- [El New England Complex Systems Institute proporciona varios recursos y documentos](#)

Los algoritmos de gráficos tienen un amplio potencial, desde prevenir el fraude y optimizar el enruteamiento de llamadas hasta predecir la propagación de la gripe. Por ejemplo, podríamos querer puntuar nodos particulares que podrían corresponder a condiciones de sobrecarga en un sistema de energía. O nos gustaría descubrir agrupaciones en el gráfico que corresponden a la congestión en un sistema de transporte.

De hecho, en 2010, los sistemas de transporte aéreo de EE. UU. Experimentaron dos eventos graves que involucraron múltiples aeropuertos congestionados que se estudiaron posteriormente mediante el análisis gráfico. Los científicos de la red P. Fleurquin, JJ Ramasco y VM Eguíluz utilizaron algoritmos gráficos para confirmar los eventos como parte de los retrasos en cascada sistemáticos y utilizan esta información para obtener consejos correctivos, como se describe en su artículo, ["Propagación sistémica de retrasos en la red de aeropuertos de Estados Unidos"](#).

Para visualizar la red que sustenta el transporte aéreo, la [figura 1-3](#) fue creada por Martin Grandjean para su artículo, ["Mundo conectado: Desenredar la red de tráfico aéreo"](#). Esta ilustración muestra claramente la estructura altamente conectada de los grupos de transporte aéreo. Muchos sistemas de transporte exhiben una distribución concentrada de enlaces con patrones claros de eje y radio que influyen en los retrasos.

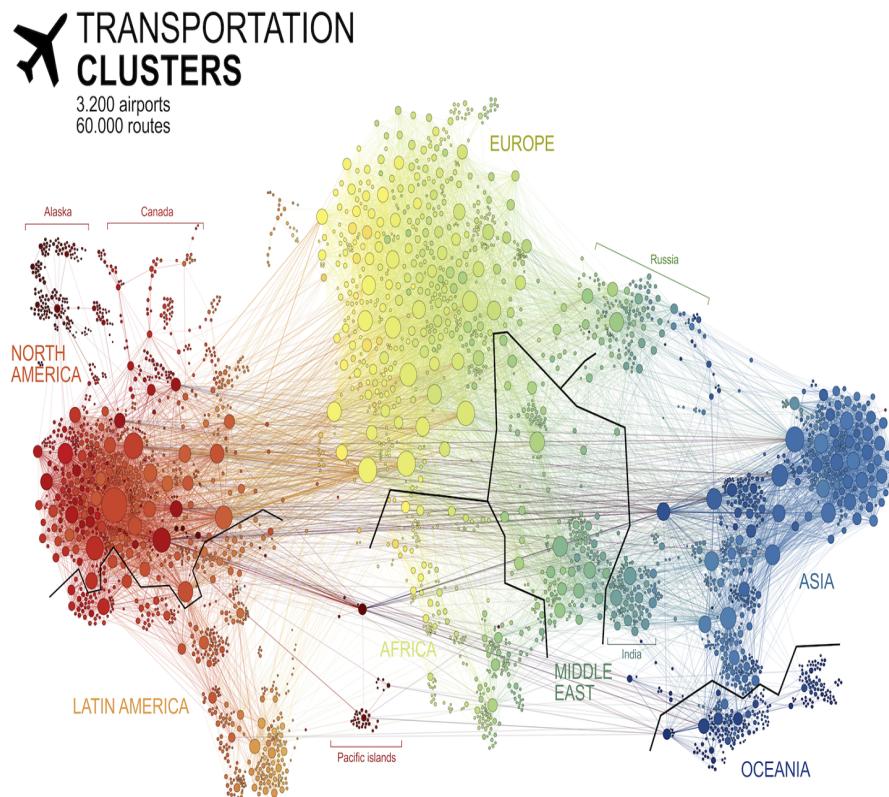


Figura 1-3. Las redes de transporte aéreo ilustran estructuras de centro y radios que evolucionan en múltiples escalas. Estas estructuras contribuyen a cómo fluye el viaje.

Los gráficos también ayudan a descubrir cómo interacciones y dinámicas muy pequeñas conducen a mutaciones globales. Unen las escalas micro y macro al representar exactamente qué cosas están interactuando dentro de las estructuras globales. Estas asociaciones se utilizan para predecir el comportamiento y determinar los enlaces que faltan. [La Figura 1-4](#) es una red alimentaria de interacciones de especies de pastizales que utilizaron análisis de gráficos para evaluar la organización jerárquica y las interacciones de especies y luego predecir las relaciones faltantes, como se detalla en el documento de A. Clauset, C. Moore y MEJ Newman, “[Estructura jerárquica y la predicción de enlaces faltantes en la red](#)” .

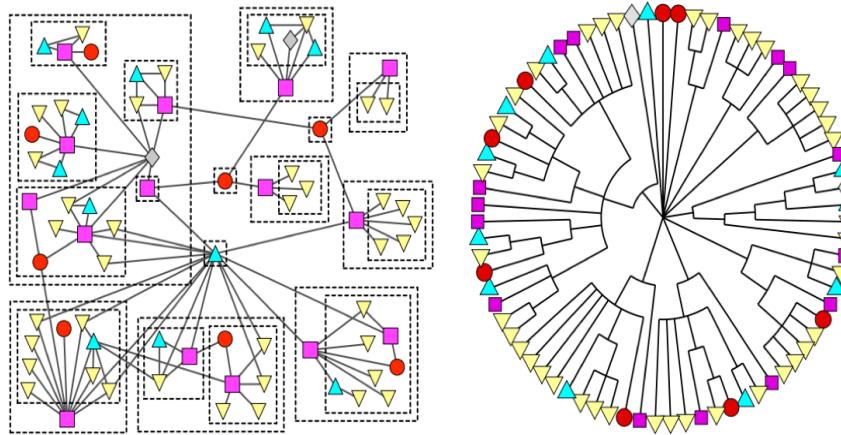


Figura 1-4. Esta red alimentaria de especies de pastizales utiliza gráficos para correlacionar las interacciones a pequeña escala con la formación de estructuras más grandes.

## Procesamiento de grafos, bases de datos, consultas y algoritmos

El procesamiento de gráficos incluye los métodos mediante los cuales se realizan las tareas y cargas de trabajo de gráficos. La mayoría de las consultas de gráficos consideran partes específicas del gráfico (por ejemplo, un nodo de inicio), y el trabajo generalmente se centra en el subgrafo que lo rodea. Este tipo de gráfico de trabajo se denomina local , e implica consultar de manera declarativa la estructura de un gráfico, como se explica en el libro [Graph Databases](#) , de Ian Robinson, Jim Webber y Emil Eifrem (O'Reilly). Este tipo de procesamiento de gráficos locales se utiliza a menudo para transacciones en tiempo real y consultas basadas en patrones.

Cuando hablamos de algoritmos de grafos, normalmente estamos buscando Patrones y estructuras globales. La entrada al algoritmo suele ser el gráfico completo, y la salida puede ser un gráfico enriquecido o algún valor agregado, como una puntuación. Clasificamos dicho procesamiento como gráfico global , e implica procesar la estructura de un gráfico utilizando algoritmos computacionales (a menudo iterativamente). Este enfoque arroja luz sobre la naturaleza general de una red a través de sus conexiones. Las organizaciones tienden a usar algoritmos de graficación para modelar sistemas y predecir el comportamiento según la forma en que se difunden las cosas, los componentes importantes, la identificación de grupos y la solidez general del sistema.

Es posible que haya algunas coincidencias en estas definiciones (a veces podemos usar el procesamiento de un algoritmo para responder a una consulta local, o viceversa), pero, de manera simplista, las operaciones de gráficos completos se procesan mediante algoritmos computacionales y las operaciones de subgrafos se consultan en las bases de datos.

Tradicionalmente, el procesamiento y análisis de transacciones se han siledeado. Esta fue una división no natural basada en limitaciones tecnológicas. Nuestra opinión es que el análisis gráfico genera transacciones más inteligentes, lo que crea nuevos datos y oportunidades para un análisis más profundo. Más recientemente, ha habido una tendencia a integrar estos silos para tomar más decisiones en tiempo real.

## OLTP y OLAP

Las operaciones de procesamiento de transacciones en línea (OLTP) suelen ser actividades cortas como reservar un boleto, acreditar una cuenta, reservar una venta, etc. OLTP implica un procesamiento voluminoso de consultas de baja latencia y una alta integridad de los datos. Aunque OLTP puede involucrar solo una pequeña cantidad de registros por transacción, los sistemas procesan muchas transacciones simultáneamente.

El procesamiento analítico en línea (OLAP) facilita consultas y análisis más complejos sobre datos históricos. Estos análisis pueden incluir múltiples fuentes de datos, formatos y tipos. Detectar tendencias, realizar escenarios hipotéticos, hacer predicciones y descubrir patrones estructurales son casos de uso típicos de OLAP. En comparación con OLTP, los sistemas OLAP procesan menos transacciones que se ejecutan durante más tiempo en muchos registros. Los sistemas OLAP están orientados hacia una lectura más rápida sin la expectativa de actualizaciones transaccionales encontradas en OLTP, y la operación orientada a lotes es común.

Recientemente, sin embargo, la línea entre OLTP y OLAP ha comenzado a difuminarse. Las aplicaciones modernas de uso intensivo de datos ahora combinan operaciones transaccionales en tiempo real con análisis. Esta combinación de procesamiento ha sido estimulada por varios avances en el software, como la administración de transacciones más escalable y el procesamiento de flujo incremental, y por hardware de memoria grande de menor costo.

Reunir análisis y transacciones permite el análisis continuo como parte natural de las operaciones regulares. A medida que se recopilan los datos, desde las máquinas de punto de venta (POS), los sistemas de fabricación o los dispositivos de Internet de las cosas (IoT), el análisis ahora admite la capacidad de hacer recomendaciones y decisiones en tiempo real mientras se procesa. Esta tendencia se observó hace varios años, y los términos para describir esta fusión incluyeron *translytics* y *Procesamiento híbrido transaccional y analítico (HTAP)*. [La Figura 1-5](#) ilustra cómo se pueden usar las réplicas de solo lectura para reunir estos diferentes tipos de procesamiento.

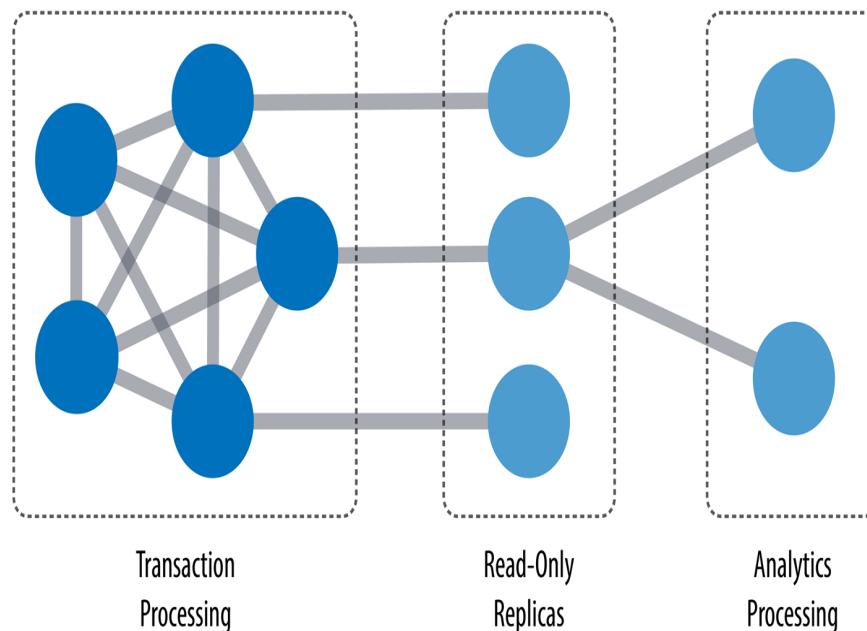


Figura 1-5. Una plataforma híbrida admite el procesamiento de consultas de baja latencia y la alta integridad de los datos requeridos para las transacciones, al tiempo que integran análisis complejos sobre grandes cantidades de datos.

Según [Gartner](#) :

[HTAP] podría redefinir la forma en que se ejecutan algunos procesos de negocios, ya que los análisis avanzados en tiempo real (por ejemplo, la planificación, la previsión y el análisis hipotético) se convierten en una parte integral del proceso en sí, en lugar de una actividad separada realizada después del hecho. Esto permitiría nuevas formas de proceso de toma de decisiones impulsado por el negocio en tiempo real. En última instancia, HTAP se convertirá en una arquitectura habilitadora clave para las operaciones comerciales inteligentes.

A medida que OLTP y OLAP se vuelven más integrados y comienzan a admitir la funcionalidad que anteriormente se ofrecía en un solo silo, ya no es necesario usar diferentes productos o sistemas de datos para estas cargas de trabajo; podemos simplificar nuestra arquitectura utilizando la misma plataforma para ambos. Esto significa que nuestras consultas analíticas pueden aprovechar los datos en tiempo real y podemos agilizar el proceso iterativo de análisis.

## ¿Por qué deberíamos preocuparnos por los algoritmos de grafos?

Los algoritmos de gráficos se utilizan para ayudar a dar sentido a los datos conectados. Vemos relaciones dentro de los sistemas del mundo real desde interacciones de proteínas a redes sociales, desde sistemas de comunicación a redes eléctricas, y desde experiencias minoristas hasta planificación de la misión en Marte. La comprensión de las redes y las conexiones dentro de ellas ofrece un increíble potencial de visión e innovación.

Los algoritmos de gráficos son especialmente adecuados para comprender estructuras y revelar patrones en conjuntos de datos que están altamente conectados. En ninguna parte es tan evidente la conectividad y la interactividad como en el big data. La cantidad de información que se ha reunido, combinado y actualizado dinámicamente es impresionante. Aquí es donde los algoritmos de grafos pueden ayudar a dar sentido a nuestros volúmenes de datos, con análisis más sofisticados que aprovechan las relaciones y mejoran la información contextual de inteligencia artificial.

A medida que nuestros datos se conectan, es cada vez más importante comprender sus relaciones e interdependencias. Los científicos que estudian el crecimiento de las redes han notado que la conectividad aumenta con el tiempo, pero no de manera uniforme. El apego preferencial es una teoría sobre cómo influye la dinámica de crecimiento en la estructura. Esta idea, ilustrada en la [Figura 1-6](#), describe la tendencia de un nodo a conectarse a otros nodos que ya tienen muchas conexiones.

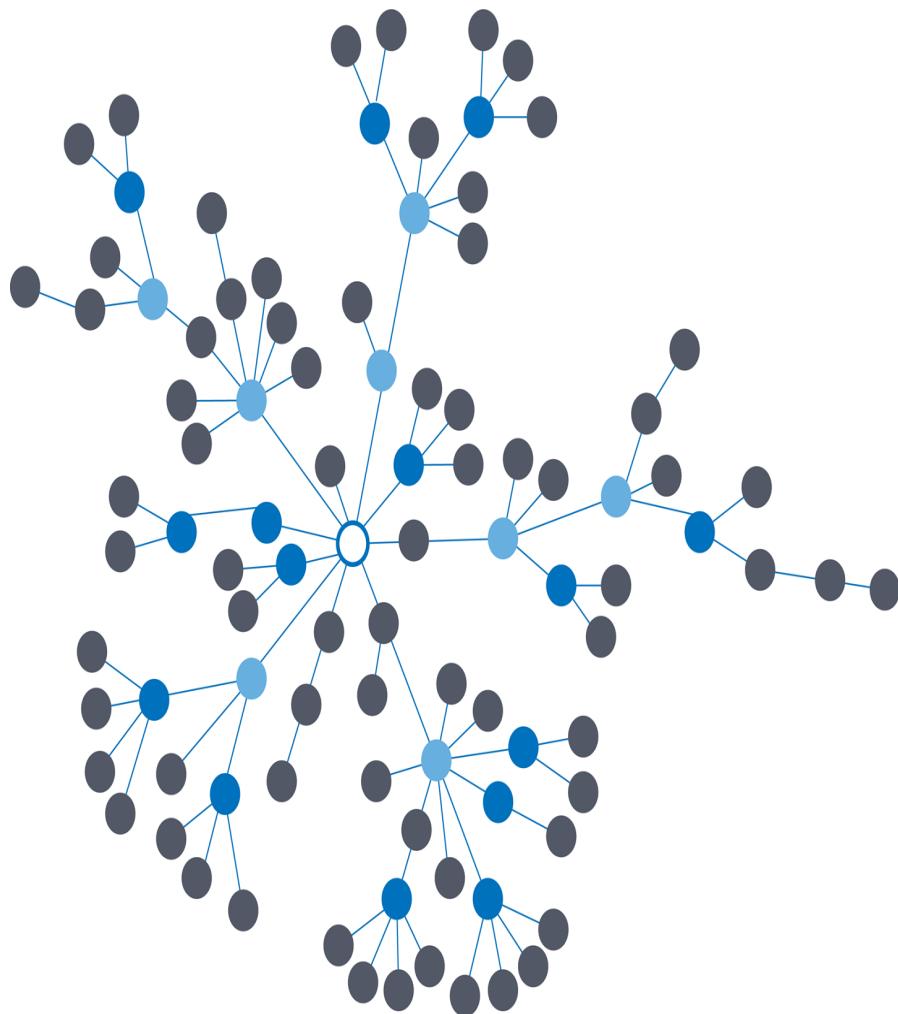


Figura 1-6. La conexión preferencial es el fenómeno en el que cuanto más conectado está un nodo, más probable es que reciba nuevos enlaces. Esto conduce a concentraciones y centros desiguales.

En su libro Sync: Cómo surge el orden del caos en el universo, la naturaleza y la vida cotidiana (Hachette), Steven Strogatz proporciona ejemplos y explica diferentes formas en que los sistemas de la vida real se autoorganizan. Independientemente de las causas subyacentes, muchos investigadores creen que la forma en que crecen las redes es inseparable de sus formas y jerarquías resultantes. Los grupos altamente densos y las redes de datos abultados tienden a desarrollarse, con una complejidad creciente junto con el tamaño de los datos. Vemos este agrupamiento de relaciones en la mayoría de las redes del mundo real en la actualidad, desde Internet hasta redes sociales como la comunidad de juegos que se muestra en la [Figura 1-7](#).

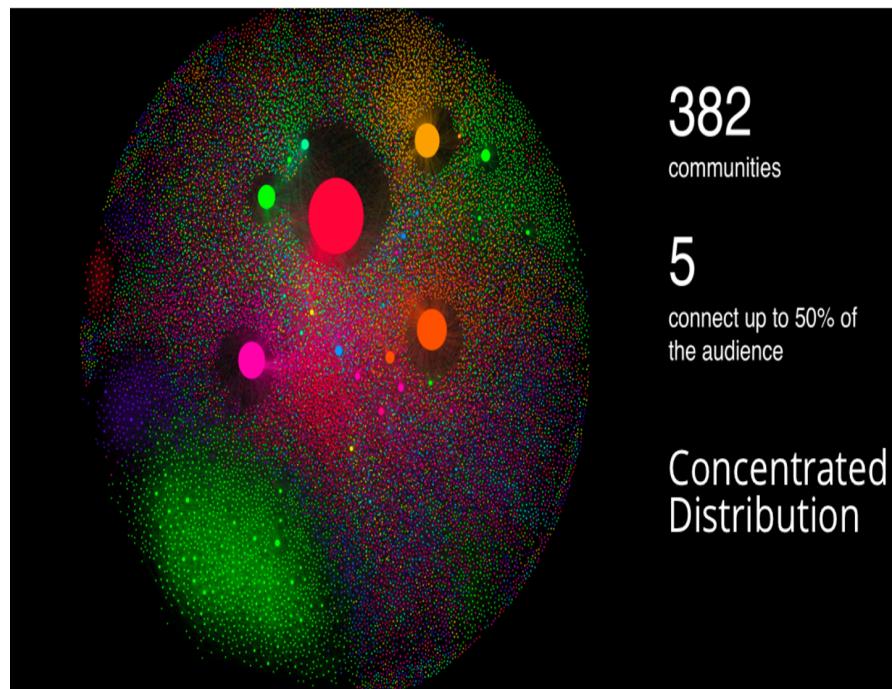


Figura 1-7. Este [análisis de la comunidad de juegos](#) muestra una concentración de conexiones en solo 5 de 382 comunidades.

El análisis de red que se muestra en la [Figura 1-7](#) fue creado por Francesco D'Orazio de Pulsar para ayudar a predecir la viralidad del contenido e informar las estrategias de distribución. D'Orazio [encontró](#) una correlación entre la concentración de la distribución de una comunidad y la velocidad de difusión de un contenido.

Esto es significativamente diferente de lo que predeciría un modelo de distribución promedio, donde la mayoría de los nodos tendrían el mismo número de conexiones. Por ejemplo, si la World Wide Web tuviera una distribución promedio de conexiones, todas las páginas tendrían aproximadamente la misma cantidad de enlaces que entran y salen. Los modelos de distribución promedio afirman que la mayoría de los nodos están conectados por igual, pero muchos tipos de gráficos y muchas redes reales muestran concentraciones. La web, en común con gráficos como viajes y redes sociales, tiene una distribución de ley de poder con unos pocos nodos altamente conectados y la mayoría de los nodos están conectados modestamente.

### LEY DE POTENCIA

Una ley de poder (también llamada ley de escala) describe la relación entre dos cantidades donde una cantidad varía como un poder de otra. Por ejemplo, el área de un cubo está relacionada con la longitud de sus lados por una potencia de 3. Un ejemplo bien conocido es la distribución de Pareto o "regla 80/20", originalmente utilizada para describir la situación en la que el 20% de una población controlaba el 80% de la riqueza. Vemos varias leyes de poder en el mundo natural y en las redes.

Tratar de "promediar" una red generalmente no funcionará bien para investigar relaciones o realizar pronósticos, porque las redes del mundo real tienen distribuciones desiguales de nodos y relaciones. Podemos ver fácilmente en la [Figura 1-8](#) cómo el uso de un promedio de características para datos desiguales podría llevar a resultados incorrectos.

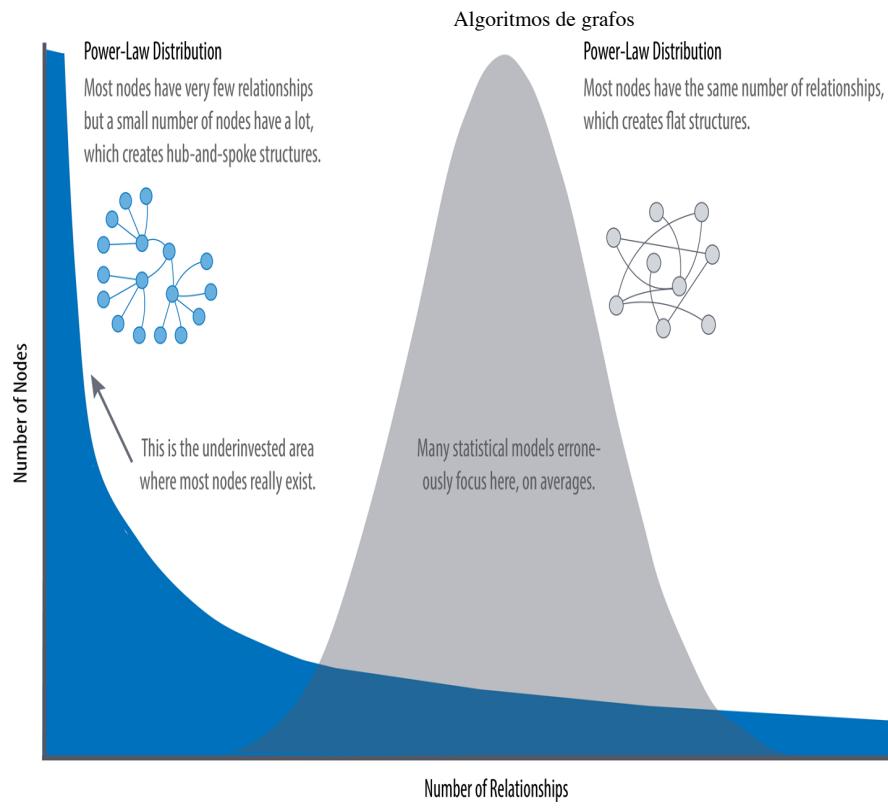


Figura 1-8. Las redes del mundo real tienen distribuciones desiguales de nodos y relaciones representadas en extremo por una distribución de ley de poder. Una distribución promedio supone que la mayoría de los nodos tienen el mismo número de relaciones y resultados en una red aleatoria.

Debido a que los datos altamente conectados no se adhieren a una distribución promedio, los científicos de redes utilizan el análisis gráfico para buscar e interpretar estructuras y distribuciones de relaciones en datos del mundo real.

No hay ninguna red en la naturaleza que sepamos que se describiría mediante el modelo de red aleatorio.

Albert-László Barabási, director del Centro de Investigación de Redes Complejas de la Universidad del Noreste, y autor de numerosos libros sobre redes científicas.

El desafío para la mayoría de los usuarios es que los datos conectados de forma densa e irregular son difíciles de analizar con las herramientas analíticas tradicionales. Puede haber una estructura allí, pero es difícil de encontrar. Es tentador adoptar un enfoque de promedios para datos desordenados, pero hacerlo ocultará patrones y garantizará que nuestros resultados no representen ningún grupo real. Por ejemplo, si promedia la información demográfica de todos sus clientes y ofrece una experiencia basada únicamente en promedios, se le garantizará que extrañará a la mayoría de las comunidades: las comunidades tienden a agruparse en torno a factores relacionados como la edad y la ocupación o el estado civil y la ubicación.

Además, el comportamiento dinámico, particularmente alrededor de eventos repentinos y ráfagas, no se puede ver con una instantánea. Para ilustrar, si imaginas un grupo social con relaciones crecientes, también esperarías más comunicaciones. Esto podría llevar a un punto de inflexión de la coordinación y una coalición posterior o, alternativamente, la formación de subgrupos y la polarización en, por ejemplo, elecciones. Se requieren métodos sofisticados para pronosticar la evolución de una red a lo largo del tiempo, pero podemos inferir el comportamiento si entendemos las estructuras e interacciones dentro de nuestros datos. El análisis gráfico se usa para predecir la resistencia del grupo debido al enfoque en las relaciones.

## Casos de uso de Graph Analytics

En el nivel más abstracto, el análisis gráfico se aplica para predecir el comportamiento y prescribir acciones para grupos dinámicos. Hacer esto requiere entender las relaciones y la estructura dentro del grupo. Los algoritmos de grafos logran esto al examinar la naturaleza general de las redes a través de sus conexiones. Con este enfoque, puede comprender la topología de los sistemas conectados y modelar sus procesos.

Hay tres grupos generales de preguntas que indican si los análisis gráficos y los algoritmos están garantizados, como se muestra en la [Figura 1-9](#) .

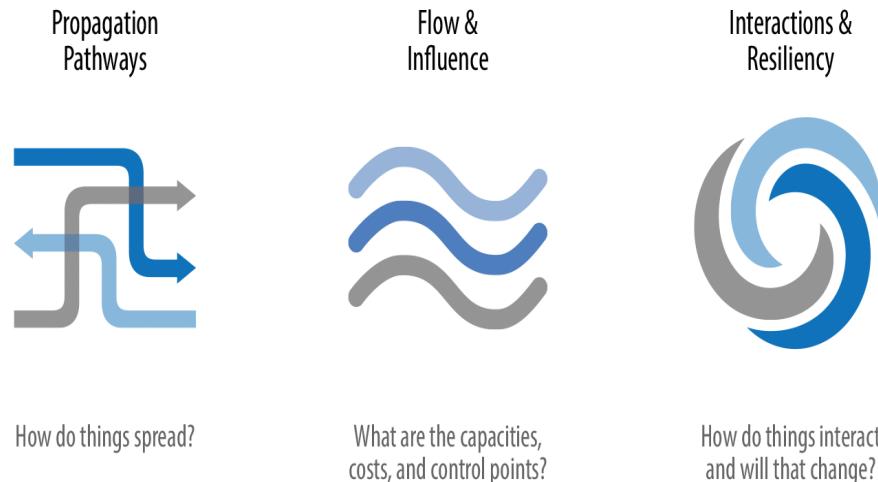


Figura 1-9. Los tipos de preguntas del análisis analítico de respuestas.

Aquí hay algunos tipos de desafíos donde se emplean algoritmos gráficos. ¿Son tus retos similares?

- Investigar la ruta de una enfermedad o una falla de transporte en cascada.
- Descubrir los componentes más vulnerables o dañinos en un ataque de red.
- Identifique la forma menos costosa o más rápida de enrutar información o recursos.
- Predecir enlaces faltantes en tus datos.
- Localizar influencia directa e indirecta en un sistema complejo.
- Descubrir jerarquías y dependencias invisibles.
- Pronostica si los grupos se fusionarán o separarán.
- Encuentre cuellos de botella o quién tiene el poder de denegar / proporcionar más recursos.
- Revelar comunidades basadas en el comportamiento para recomendaciones personalizadas.
- Reducir los falsos positivos en el fraude y detección de anomalías.
- Extraer más características predictivas para el aprendizaje automático.

## Conclusión

En este capítulo, hemos visto cómo los datos de hoy están extremadamente conectados y las implicaciones de esto. Existen prácticas científicas sólidas para el análisis de las dinámicas de grupo y las relaciones, pero esas herramientas no siempre son comunes en las empresas. A medida que evaluamos técnicas analíticas avanzadas, debemos considerar la naturaleza de nuestros datos y si necesitamos entender los atributos de la comunidad o predecir comportamientos complejos. Si nuestros datos representan una red, debemos evitar la tentación de reducir los factores a un promedio. En su lugar, deberíamos usar herramientas que coincidan con nuestros datos y los conocimientos que estamos buscando.

En el siguiente capítulo, cubriremos conceptos y terminología de gráficos.

# Capítulo 2. Teoría y conceptos de grafos.

En este capítulo, establecemos el marco y cubrimos la terminología de los algoritmos gráficos. Se explican los conceptos básicos de la teoría de grafos, con un enfoque en los conceptos que son más relevantes para un profesional.

Describiremos cómo se representan los gráficos y luego explicaremos los diferentes tipos de gráficos y sus atributos. Esto será importante más adelante, ya que las características de nuestro gráfico informarán nuestras elecciones de algoritmos y nos ayudarán a interpretar los resultados. Terminaremos el capítulo con una descripción general de los tipos de algoritmos de gráficos que se detallan en este libro.

## Terminología

El gráfico de propiedades etiquetadas es una de las formas más populares de modelar datos de gráficos.

Una etiqueta marca un nodo como parte de un grupo. En la [Figura 2-1](#), tenemos dos grupos de nodos: Persona y Coche. (Aunque en la teoría de gráficos clásica una etiqueta se aplica a un solo nodo, ahora se usa comúnmente para referirse a un grupo de nodos).

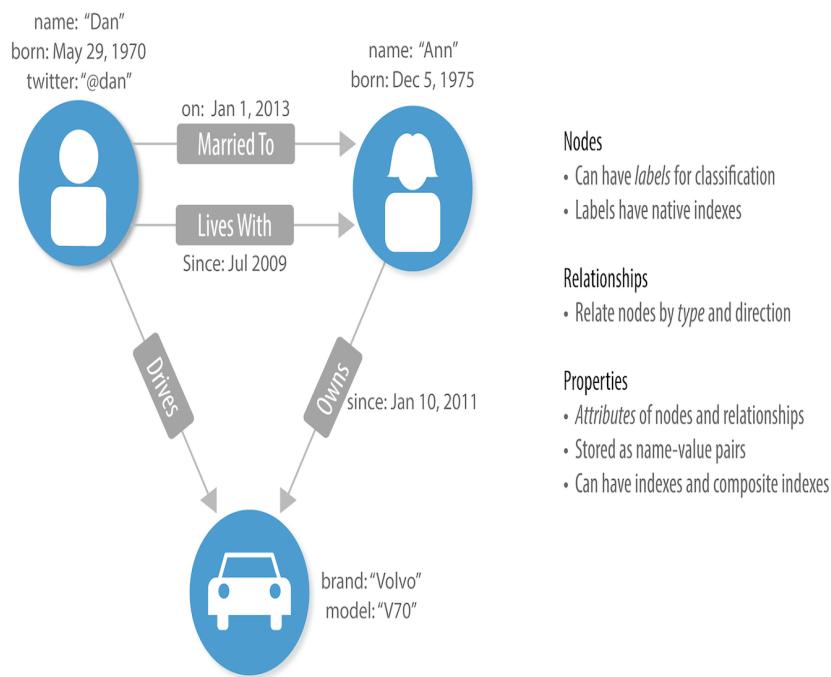


Figura 2-1. Un modelo de gráfico de propiedad etiquetada es una forma flexible y concisa de representar datos conectados.

Las relaciones se clasifican según el tipo de relación. Nuestro ejemplo incluye los tipos de relación de DRIVES, OWNS, LIVES\_WITH y MARRIED\_TO.

Las propiedades son sinónimo de atributos y pueden contener una variedad de tipos de datos, desde números y cadenas hasta datos espaciales y temporales. En la [Figura 2-1](#) asignamos las propiedades como pares de nombre-valor, donde el nombre de la propiedad aparece primero y luego su valor. Por ejemplo, el nodo Persona a la izquierda tiene un nombre de propiedad: "Dan", y la relación MARRIED\_TO tiene una propiedad de: 1 de enero de 2013.

Un subgrafo es un grafico dentro de un grafo mas grande. Los subgrafos son útiles como filtros, como cuando necesitamos un subconjunto con características particulares para un análisis centrado.

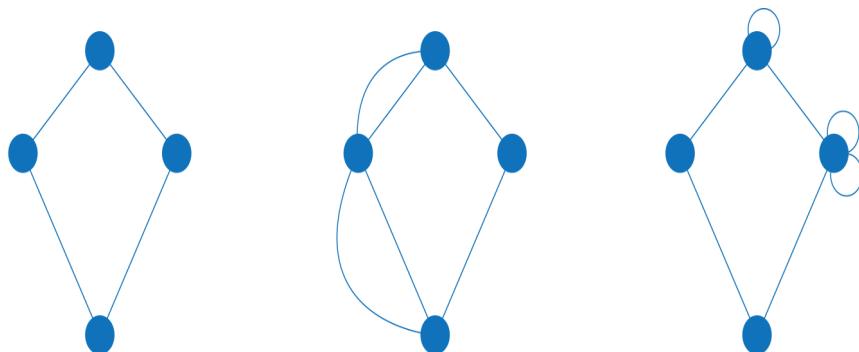
Un camino es un grupo de nodos y sus relaciones de conexión. Un ejemplo de una ruta simple, basada en la [Figura 2-1](#), podría contener los nodos Dan, Ann y Car y las relaciones DRIVES y OWNS.

Los gráficos varían en tipo, forma y tamaño, así como el tipo de atributos que se pueden usar para el análisis. A continuación, describiremos los tipos de gráficos más adecuados para los algoritmos de gráficos. Tenga en

cuenta que estas explicaciones se aplican tanto a los gráficos como a los subgrafos.

## Tipos de grafos y estructuras

En la teoría de gráficos clásica, el término gráfico se equipara con un gráfico simple (o estricto) donde los nodos solo tienen una relación entre ellos, como se muestra en el lado izquierdo de la [Figura 2-2](#). Sin embargo, la mayoría de los gráficos del mundo real tienen muchas relaciones entre nodos e incluso relaciones de autorreferencia. Hoy en día, este término se usa comúnmente para los tres tipos de gráficosEn la [Figura 2-2](#), también usamos el término de manera inclusiva.



**Simple Graph**  
Node pairs can only have one relationship between them.

**Multigraph**  
Node pairs can have multiple relationships between them.

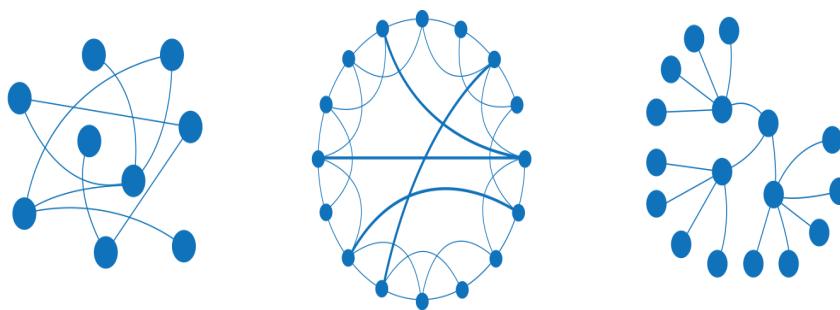
**Graph (also Pseudograph)**  
Node pairs can have multiple relationships between them.  
Nodes can loop back to themselves.

Figura 2-2. En este libro, usamos el término gráfico para incluir cualquiera de estos tipos clásicos de gráficos.

## Estructuras aleatorias, de mundo pequeño, sin escala

Los gráficos adquieren muchas formas. [La Figura 2-3](#) muestra tres tipos de redes representativas:

- Redes aleatorias
- Redes del mundo pequeño.
- Redes sin escala



**Random**  
Average distributions.  
No structure or hierachal patterns.

**Small-World**  
High local clustering and short average path lengths.  
Hub-and-spoke architecture.

**Scale-Free**  
Hub-and-spoke architecture preserved at multiple scales.  
High power law distribution.

Figura 2-3. Tres estructuras de red con gráficos y comportamientos distintivos.

- En una distribución de conexiones completamente promedio, se forma una red aleatoria sin jerarquías. Este tipo de gráfico sin forma es "plano" sin patrones discernibles. Todos los nodos tienen la misma probabilidad de estar conectados a cualquier otro nodo.

- Una red de mundo pequeño es extremadamente común en las redes sociales; muestra conexiones localizadas y algunos patrones de hub-y-radios. El juego "[Seis grados de Kevin Bacon](#)" podría ser el ejemplo más conocido del efecto del mundo pequeño. Aunque te asocias principalmente con un pequeño grupo de amigos, nunca estás a muchos saltos de distancia de cualquier otra persona, incluso si son un actor famoso o al otro lado del planeta.
- Se produce una red sin escala cuando hay distribuciones de ley de energía y se conserva una arquitectura de centro y radio independientemente de la escala, como en la World Wide Web.

Estos tipos de red producen gráficos con estructuras, distribuciones y comportamientos distintivos. Mientras trabajamos con algoritmos de grafos, llegaremos a reconocer patrones similares en nuestros resultados.

## Sabores de grafos

Para aprovechar al máximo los algoritmos de gráficos, es importante que nos familiaricemos con los gráficos más característicos que encontraremos. [La tabla 2-1](#) resume los atributos comunes del gráfico. En las siguientes secciones veremos los diferentes sabores con más detalle.

Tabla 2-1. Atributos comunes de los gráficos.

Atributo gráfico	Factor clave	Consideración del algoritmo
Conectado versus desconectado	Si hay una ruta entre dos nodos en el gráfico, independientemente de la distancia	Las islas de nodos pueden causar un comportamiento inesperado, como quedarse atascado o no poder procesar los componentes desconectados.
Ponderado versus no ponderado	Si hay valores (específicos del dominio) en relaciones o nodos	Muchos algoritmos esperan ponderaciones y veremos diferencias significativas en el rendimiento y los resultados cuando se ignoran.
Dirigido versus no dirigido	Si las relaciones definen explícitamente un nodo de inicio y fin	Esto agrega un rico contexto para inferir un significado adicional. En algunos algoritmos, puede establecer explícitamente el uso de uno, ambos o ninguna dirección.
Cíclico versus acíclico	Si las rutas comienzan y terminan en el mismo nodo	Los gráficos cíclicos son comunes, pero los algoritmos deben ser cuidadosos (generalmente almacenando el estado transversal) o los ciclos pueden evitar la terminación. Los gráficos acíclicos (o árboles en expansión) son la base de muchos algoritmos de gráficos.
Escasa versus densa	Relación relación de nodo	Las gráficas extremadamente densas o muy escasamente conectadas pueden causar resultados divergentes. El modelado de datos puede ayudar, asumiendo que el dominio no es inherentemente denso o disperso.
Monopartita, bipartita y k-partita	Si los nodos se conectan solo a otro tipo de nodo (p. Ej., A los usuarios les gustan las películas) o a muchos otros tipos de nodos (p. Ej., A los usuarios les gustan las películas)	Útil para crear relaciones para analizar y proyectar gráficos más útiles.

## Gráficos conectados y desconectados

Un gráfico está conectado si hay una ruta entre todos los nodos. Si tenemos islas en nuestro gráfico, está desconectado. Si los nodos en esas islas están conectados, se denominan componentes (o, a veces, agrupaciones), como se muestra en la [Figura 2-4](#).

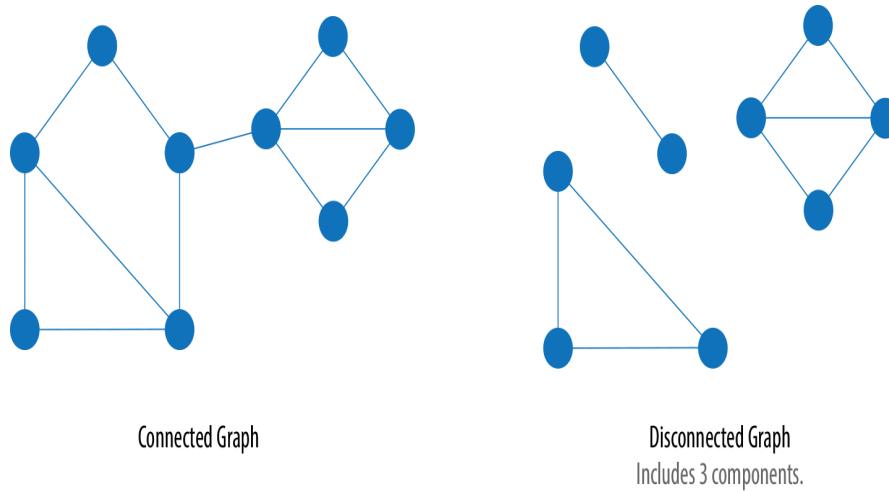


Figura 2-4. Si tenemos islas en nuestro gráfico, es un gráfico desconectado.

Algunos algoritmos luchan con gráficos desconectados y pueden producir resultados engañosos. Si tenemos resultados inesperados, un buen primer paso es verificar la estructura de nuestro gráfico.

## Gráficos no ponderados versus gráficos ponderados

Los gráficos no ponderados no tienen valores de peso asignados a sus nodos o relaciones. Para los gráficos ponderados, estos valores pueden representar una variedad de medidas, como costo, tiempo, distancia, capacidad o incluso una priorización específica del dominio. [La figura 2-5](#) visualiza la diferencia.

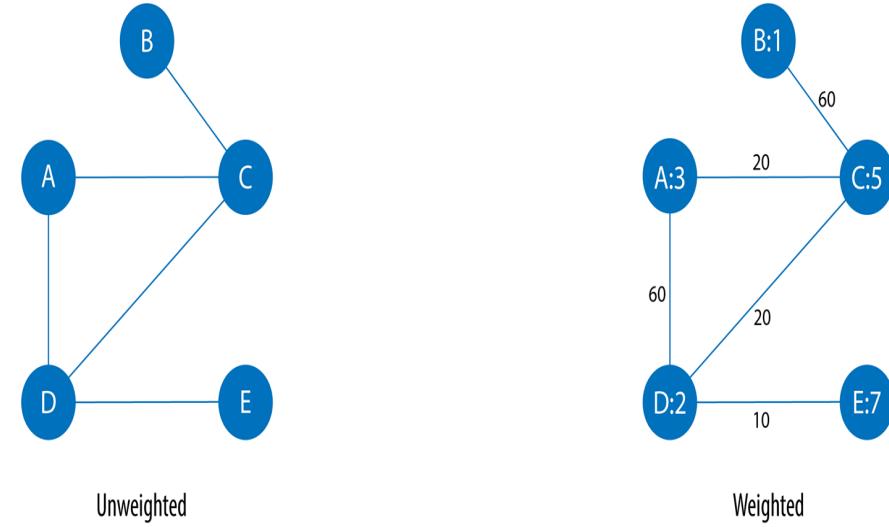


Figura 2-5. Los gráficos ponderados pueden contener valores en relaciones o nodos.

Los algoritmos básicos de grafos pueden usar pesos para el procesamiento como una representación de la fuerza o el valor de las relaciones. Muchos algoritmos calculan métricas que luego se pueden usar como ponderaciones para el procesamiento de seguimiento. Algunos algoritmos actualizan los valores de peso a medida que avanzan para encontrar totales acumulativos, valores más bajos u óptimos.

Un uso clásico para gráficos ponderados es en algoritmos de búsqueda de caminos. Dichos algoritmos respaldan las aplicaciones de mapeo en nuestros teléfonos y computan las rutas de transporte más cortas / más baratas / más rápidas entre ubicaciones. Por ejemplo, la [Figura 2-6](#) utiliza dos métodos diferentes para calcular la ruta más corta.

## What's the Shortest Path Between A and E?

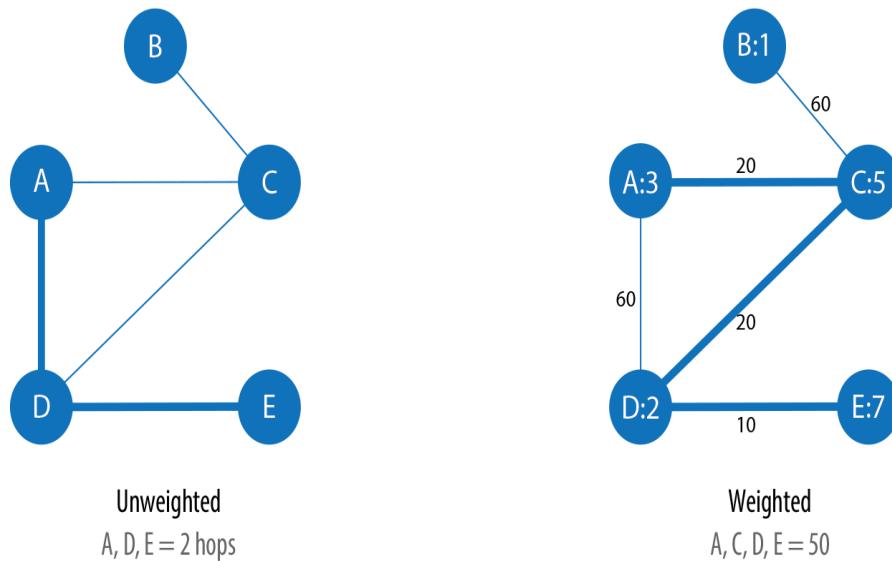


Figura 2-6. Las rutas más cortas pueden variar para gráficos ponderados y no ponderados idénticos.

Sin ponderaciones, nuestra ruta más corta se calcula en términos del número de relaciones (comúnmente llamadas saltos). A y E tienen una ruta más corta de dos saltos, lo que indica que solo hay un nodo (D) entre ellos. Sin embargo, la ruta ponderada más corta de A a E nos lleva de A a C a de D a E. Si los pesos representan una distancia física en kilómetros, la distancia total sería de 50 km. En este caso, la ruta más corta en términos de número de saltos equivaldría a una ruta física más larga de 70 km.

## Gráficos no direccionalizados versus gráficos dirigidos

En un gráfico no dirigido, las relaciones se consideran bidireccionales (por ejemplo, amistades). En una gráfica dirigida, las relaciones tienen una dirección específica. Las relaciones que apuntan a un nodo se conocen como enlaces internos y, como era de esperar, los enlaces externos son aquellos que se originan en un nodo.

La dirección añade otra dimensión de la información. Las relaciones del mismo tipo pero en direcciones opuestas tienen un significado semántico diferente, expresando una dependencia o indicando un flujo. Esto puede ser utilizado como un indicador de credibilidad o fuerza de grupo. Las preferencias personales y las relaciones sociales se expresan muy bien con la dirección.

Por ejemplo, si asumimos en la [Figura 2-7](#) que el gráfico dirigido era una red de estudiantes y que las relaciones eran "me gusta", entonces calculamos que A y C son más populares.

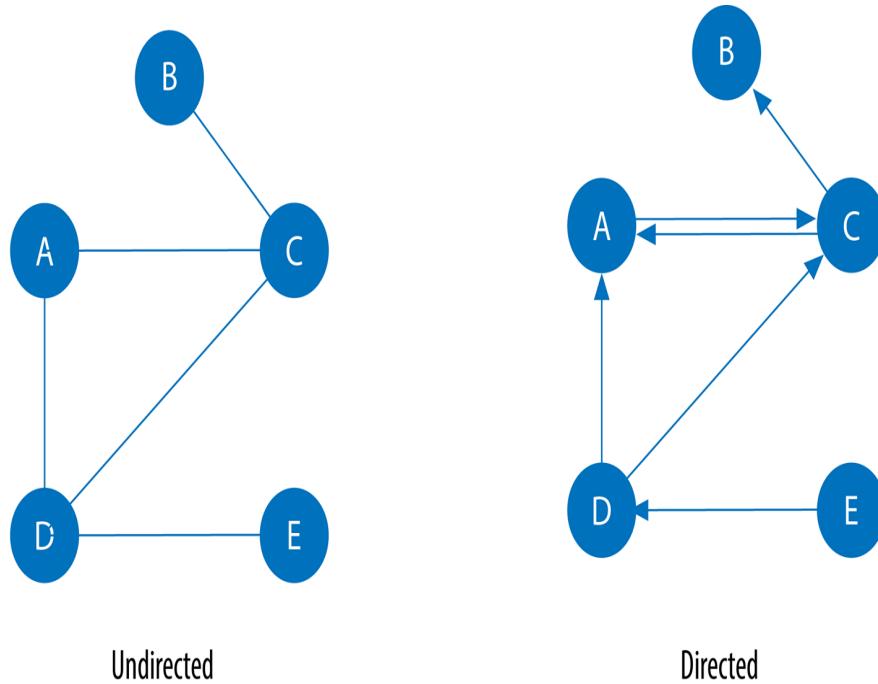


Figura 2-7. Muchos algoritmos nos permiten calcular solo en base a conexiones entrantes o salientes, en ambas direcciones o sin dirección.

Las redes de carreteras ilustran por qué podríamos usar ambos tipos de gráficos. Por ejemplo, las carreteras entre ciudades a menudo se recorren en ambas direcciones. Sin embargo, dentro de las ciudades, algunas carreteras son calles de un solo sentido. (¡Lo mismo es cierto para algunos flujos de información!)

Obtenemos diferentes resultados ejecutando algoritmos de forma no dirigida en comparación con los dirigidos. En un gráfico no dirigido, por ejemplo, para carreteras o amistades, asumiríamos que todas las relaciones siempre van en ambos sentidos.

Si reimaginamos la [Figura 2-7](#) como una red de carreteras dirigida, puede conducir a A desde C y D, pero solo puede salir por C. Además, si no hubiera relaciones de A a C, eso indicaría un callejón sin salida. Tal vez sea menos probable para una red vial de una sola vía, pero no para un proceso o una página web.

## Gráficos acíclicos versus gráficos cíclicos

En teoría de grafos, los ciclos son rutas a través de relaciones y nodos que comienzan y terminan en el mismo nodo. Una gráfica acíclica no tiene tales ciclos. Como se muestra en la [Figura 2-8](#), tanto los gráficos dirigidos como los no dirigidos pueden tener ciclos, pero cuando se dirigen, los caminos siguen la dirección de la relación. Un gráfico acíclico dirigido (DAG), que se muestra en el Gráfico 1, por definición siempre tendrá callejones sin salida (también llamados nodos de hoja).

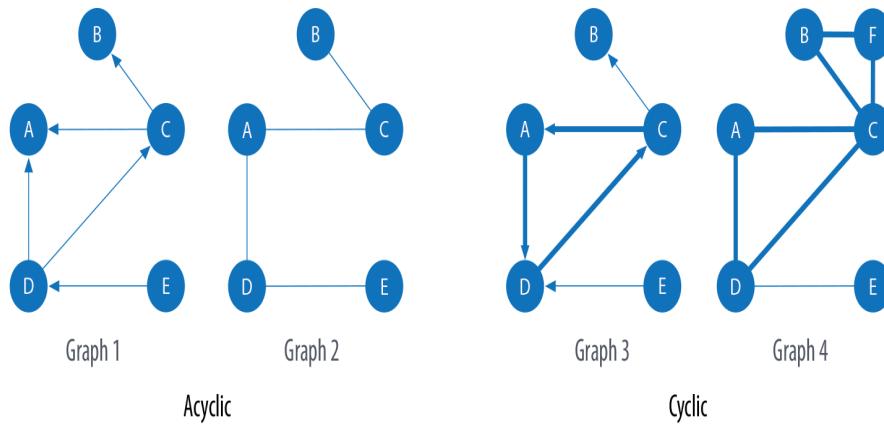


Figura 2-8. En los gráficos acíclicos, es imposible comenzar y terminar en el mismo nodo sin volver sobre nuestros pasos.

Las gráficas 1 y 2 no tienen ciclos, ya que no hay manera de comenzar y terminar en el mismo nodo sin repetir una relación. ¡Quizás recuerdes del [Capítulo 1](#) que no repetir relaciones fue el problema de los puentes

de Königsberg que inició la teoría de grafos! El gráfico 3 en la [Figura 2-8](#) muestra un ciclo simple después de ADCA sin nodos repetidos. En el Gráfico 4, el gráfico cíclico no dirigido se ha vuelto más interesante al agregar un nodo y una relación. Ahora hay un ciclo cerrado con un nodo repetido (C), después de BFCDACB. En realidad, hay varios ciclos en el gráfico 4.

Los ciclos son comunes y, a veces, necesitamos convertir gráficos cíclicos en gráficos acíclicos (al cortar las relaciones) para eliminar los problemas de procesamiento. Los gráficos acíclicos dirigidos surgen naturalmente en los historiales de programación, genealogía y versiones.

## Arboles

En la teoría de gráficos clásica, un gráfico acíclico que no está dirigido se llama **árbol**. En informática, los árboles también pueden ser dirigidos. Una definición más inclusiva sería un gráfico donde dos nodos están conectados por una sola ruta. Los árboles son significativos para entender las estructuras de grafos y muchos algoritmos. Desempeñan un papel clave en el diseño de redes, estructuras de datos y optimizaciones de búsqueda para mejorar la categorización o las jerarquías organizacionales.

Mucho se ha escrito sobre los árboles y sus variaciones. [La Figura 2-9](#) ilustra los árboles comunes que probablemente encontraremos.

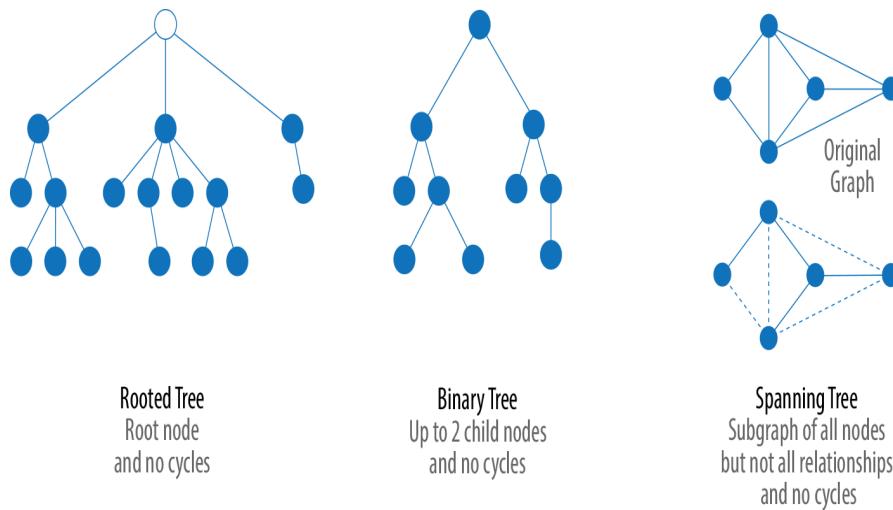


Figura 2-9. De estos gráficos de árboles prototípicos, los árboles de distribución se utilizan con mayor frecuencia para los algoritmos de graficos.

De estas variaciones, los árboles abarcadores son los más relevantes para este libro. Un árbol de expansión es un subgrafo que incluye todos los nodos de un grafo acíclico más grande pero no todas las relaciones. Un árbol de expansión mínimo conecta todos los nodos de una gráfica con el menor número de saltos o las rutas menos ponderadas.

## Gráficos escasos frente a gráficos densos

La dispersión de una gráfica se basa en el número de relaciones que tiene en comparación con el número máximo posible de relaciones, lo que ocurriría si hubiera una relación entre cada par de nodos. Un gráfico donde cada nodo tiene una relación con cada otro nodo se llama un gráfico completo, o una camarilla para los componentes. Por ejemplo, si todos mis amigos se conocieran, sería una pandilla.

La densidad máxima de un gráfico es el número de relaciones posibles en un gráfico completo. Se calcula con la fórmula  $M = \frac{N(N-1)}{2}$  donde N es el número de nodos. Para medir la densidad real usamos la fórmula  $D = \frac{R}{N(N-1)}$  donde R es el número de relaciones. En la [Figura 2-10](#), podemos ver tres medidas de densidad real para gráficos no dirigidos.

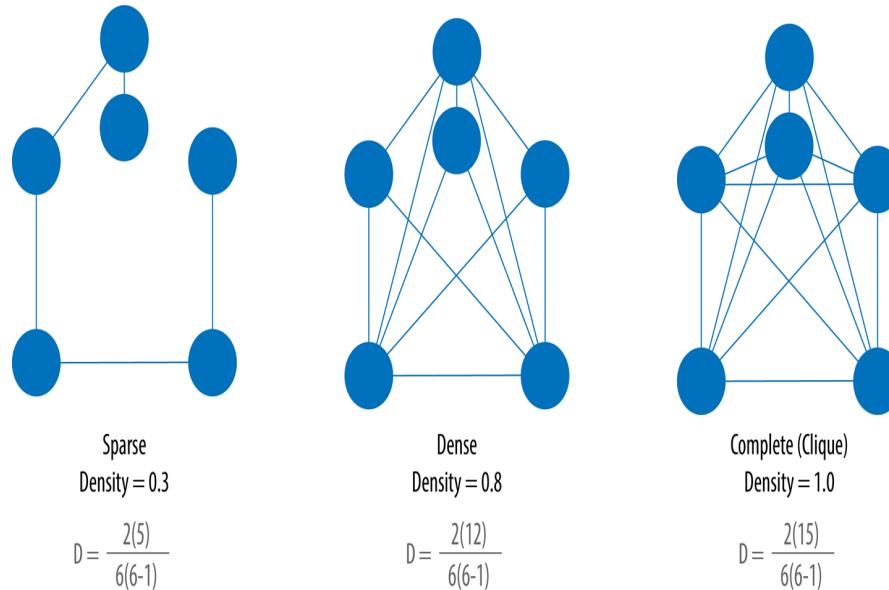


Figura 2-10. Verificar la densidad de una gráfica puede ayudarlo a evaluar resultados inesperados.

Aunque no hay una línea divisoria estricta, cualquier gráfico con una densidad real que se aproxime a la densidad máxima se considera denso. La mayoría de los gráficos basados en redes reales tienden a ser escasos, con una correlación aproximadamente lineal de nodos totales a relaciones totales. Este es especialmente el caso en el que entran en juego elementos físicos, como las limitaciones prácticas de cuántos alambres, tuberías, carreteras o amistades puede unir en un punto.

Algunos algoritmos devolverán resultados sin sentido cuando se ejecuten en gráficos extremadamente dispersos o densos. Si un gráfico es demasiado escaso, puede que no haya suficientes relaciones para que los algoritmos calculen resultados útiles. Alternativamente, los nodos muy densamente conectados no agregan mucha información adicional ya que están muy conectados. Las altas densidades también pueden sesgar algunos resultados o agregar complejidad computacional. En estas situaciones, filtrar el subgrafo relevante es un enfoque práctico.

## Gráficos monopartitos, bipartitos y k-partitas

La mayoría de las redes contienen datos con varios tipos de nodos y relaciones. Los algoritmos gráficos, sin embargo, a menudo consideran solo un tipo de nodo y un tipo de relación. Los gráficos con un tipo de nodo y tipo de relación a veces se denominan monopartitos.

Un gráfico bipartito es un gráfico cuyos nodos se pueden dividir en dos conjuntos, de modo que las relaciones solo conectan un nodo de un conjunto a un nodo de un conjunto diferente. [La Figura 2-11](#) muestra un ejemplo de un gráfico de este tipo. Tiene dos conjuntos de nodos: un conjunto de espectadores y un conjunto de programas de televisión. Solo hay relaciones entre los dos conjuntos y no hay conexiones dentro del conjunto. En otras palabras, en el Gráfico 1, los programas de televisión solo están relacionados con los espectadores, no con otros programas de televisión, y los espectadores tampoco están directamente vinculados a otros espectadores.

A partir de nuestro gráfico bipartito de espectadores y programas de televisión, creamos dos proyecciones monopartitas: el gráfico 2 de conexiones de espectadores basados en programas en común y el gráfico 3 de programas de televisión basados en espectadores en común. También podemos filtrar según el tipo de relación, como observado, calificado o revisado.

Proyectar gráficos monopartitos con conexiones inferidas es una parte importante del análisis de gráficos. Estos tipos de proyecciones ayudan a descubrir relaciones y cualidades indirectas. Por ejemplo, en el Gráfico 2 en la [Figura 2-11](#), Bev y Ann han visto solo un programa de televisión en común, mientras que Bev y Evan tienen dos programas en común. En el Gráfico 3 hemos ponderado las relaciones entre los programas de televisión por las vistas agregadas de los espectadores en común. Esta, u otras métricas como la similitud, pueden usarse para inferir el significado entre actividades como ver Battlestar Galactica y Firefly. Eso puede

informar nuestra recomendación para alguien similar a Evan que, en la [Figura 2-11](#), acaba de terminar de ver el último episodio de Firefly .

Los gráficos de k-partite hacen referencia al número de tipos de nodo que tienen nuestros datos ( k ). Por ejemplo, si tenemos tres tipos de nodos, tendríamos un gráfico tripartito. Esto solo extiende los conceptos bipartitos y monopartitos para dar cuenta de más tipos de nodos. Muchas gráficas del mundo real, especialmente las gráficas de conocimiento, tienen un gran valor para k, ya que combinan muchos conceptos y tipos diferentes de información. Un ejemplo del uso de un número mayor de tipos de nodos es crear nuevas recetas al asignar un conjunto de recetas a un conjunto de ingredientes a un compuesto químico, y luego deducir nuevas mezclas que conecten las preferencias populares. También podríamos reducir el número de tipos de nodos por generalización, como el tratamiento de muchas formas de nodos, como espinacas o coles, como "verdes de hoja verde".

Ahora que hemos revisado los tipos de gráficos con los que es más probable que trabajemos, aprendamos sobre los tipos de algoritmos de gráficos que ejecutaremos en esos gráficos.

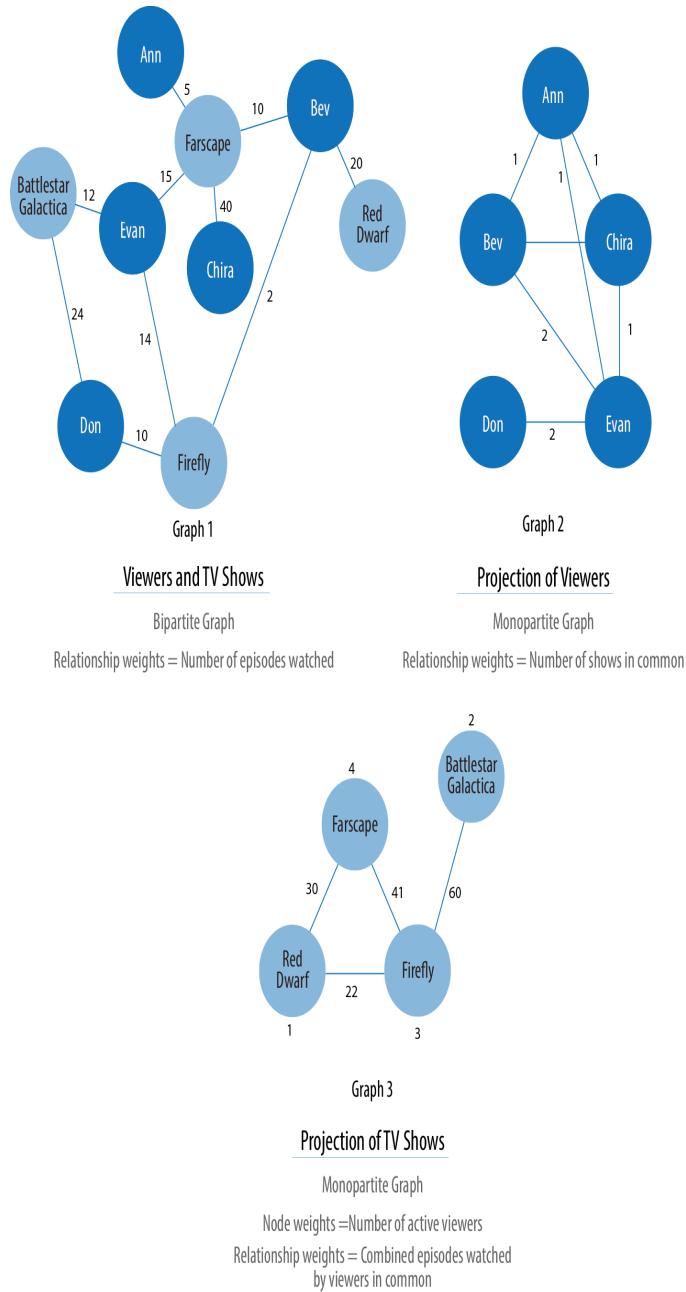


Figura 2-11. Los gráficos bipartitos a menudo se proyectan en gráficos monopartitos para un análisis más específico.

## Tipos de algoritmos de grafos

Veamos las tres áreas de análisis que están en el corazón de los algoritmos de gráficos. Estas categorías corresponden a los capítulos sobre algoritmos para búsqueda de rutas y búsqueda, cálculo de centralidad y detección de comunidades.

## Pathfinding

Las rutas son fundamentales para el análisis gráfico y los algoritmos, por lo que aquí es donde comenzaremos nuestros capítulos con ejemplos de algoritmos específicos. La búsqueda de rutas más cortas es probablemente la tarea más frecuente realizada con algoritmos de grafos y es un precursor de varios tipos diferentes de análisis. La ruta más corta es la ruta transversal con menos saltos o menor peso. Si el gráfico está dirigido, entonces es la ruta más corta entre dos nodos según lo permitido por las direcciones de la relación.

### TIPOS DE RUTA

La ruta más corta promedio se usa para considerar la eficiencia general y la capacidad de recuperación de las redes, como comprender la distancia promedio entre las estaciones de metro. Algunas veces también podemos querer entender la ruta más larga y optimizada para situaciones como determinar qué estaciones de metro están más alejadas o tener la mayor cantidad de paradas entre ellas, incluso cuando se elige la mejor ruta. En este caso, usaríamos el diámetro de una gráfica para encontrar la ruta más larga y corta entre todos los pares de nodos.

## Centralidad

La centralidad tiene que ver con comprender qué nodos son más importantes en una red. ¿Pero qué entendemos por importancia? Existen diferentes tipos de algoritmos de centralidad creados para medir diferentes cosas, como la capacidad de difundir información rápidamente en lugar de unir grupos distintos. En este libro, nos centraremos en cómo se estructuran los nodos y las relaciones.

## Detección de la comunidad

La conectividad es un concepto central de la teoría de grafos que permite un análisis de red sofisticado como la búsqueda de comunidades. La mayoría de las redes del mundo real exhiben subestructuras (a menudo casi fractales) de subgrafos más o menos independientes.

La conectividad se utiliza para encontrar comunidades y cuantificar la calidad de las agrupaciones. La evaluación de diferentes tipos de comunidades dentro de un gráfico puede descubrir estructuras, como centros y jerarquías, y tendencias de los grupos para atraer o repeler a otros. Estas técnicas se utilizan para estudiar fenómenos emergentes como los que conducen a cámaras de eco y efectos de burbujas de filtro.

## Resumen

Los gráficos son intuitivos. Se alinean con cómo pensamos y dibujamos sistemas. Los principios básicos del trabajo con gráficos se pueden asimilar rápidamente una vez que hemos desentrañado algunos de los términos y capas. En este capítulo, hemos explicado las ideas y expresiones utilizadas más adelante en este libro y describimos los tipos de gráficos que encontrará.

### REFERENCIAS DE LA TEORÍA DE GRAFOS

Si está emocionado de aprender más sobre la teoría de los gráficos, hay algunos textos introductorios que recomendamos:

- Introducción a la teoría de los gráficos , por Richard J. Trudeau (Dover), es una introducción muy bien escrita y suave.
- Introducción a Graph Theory , Fifth Ed., De Robin J. Wilson (Pearson), es una introducción sólida con buenas ilustraciones.
- Teoría de grafos y sus aplicaciones , Tercera edición, de Jonathan L. Gross, Jay Yellen y Mark Anderson (Chapman y Hall), asume más antecedentes matemáticos y proporciona más detalles y ejercicios.

A continuación, veremos el procesamiento de gráficos y los tipos de análisis antes de profundizar en cómo usar los algoritmos de gráficos en Apache Spark y Neo4j.

# Capítulo 3. Plataformas gráficas y procesamiento.

---

En este capítulo, cubriremos rápidamente diferentes métodos para el procesamiento de gráficos y los enfoques de plataforma más comunes. Examinaremos más de cerca las dos plataformas utilizadas en este libro, Apache Spark y Neo4j, y cuándo pueden ser adecuadas para diferentes requisitos. Se incluyen pautas de instalación de la plataforma para prepararlo para los siguientes capítulos.

## Plataforma gráfica y consideraciones de procesamiento

El procesamiento analítico de gráficos tiene cualidades únicas, como el cálculo que es impulsado por la estructura, enfocado globalmente y difícil de analizar. En esta sección veremos las consideraciones generales para las plataformas gráficas y el procesamiento.

## Consideraciones de la plataforma

Existe un debate sobre si es mejor escalar o escalar el procesamiento de gráficos. ¿Debería usar potentes máquinas multinúcleo y de gran memoria y concentrarse en estructuras de datos eficientes y algoritmos de multiproceso? ¿O bien valen la pena las inversiones en marcos de procesamiento distribuido y algoritmos relacionados?

Un enfoque de evaluación útil es la configuración que supera a un solo hilo (COST), como se describe en el documento de investigación "["Escalabilidad! Pero, ¿a qué costo?"](#)" Por F. McSherry, M. Isard y D. Murray. COST nos proporciona una forma de comparar la escalabilidad de un sistema con la sobrecarga que introduce el sistema. El concepto central es que un sistema bien configurado que utiliza un algoritmo optimizado y una estructura de datos puede superar a las actuales soluciones de escala general de propósito general. Es un método para medir las ganancias de rendimiento sin recompensar los sistemas que ocultan las ineficiencias a través de la paralelización. Separar las ideas de escalabilidad y uso eficiente de los recursos nos ayudará a construir una plataforma configurada explícitamente para nuestras necesidades.

Algunos enfoques de las plataformas gráficas incluyen soluciones altamente integradas que optimizan los algoritmos, el procesamiento y la recuperación de memoria para trabajar en una coordinación más estrecha.

## Consideraciones de procesamiento

Existen diferentes enfoques para expresar el procesamiento de datos; por ejemplo, procesamiento de flujo o por lotes o el paradigma de reducción de mapas para datos basados en registros. Sin embargo, para los datos de gráficos, también existen enfoques que incorporan las dependencias de datos inherentes a las estructuras de gráficos en su procesamiento:

### Centrado en el nodo

Este enfoque utiliza los nodos como unidades de procesamiento, haciéndolos acumular y calcular el estado y comunicar los cambios de estado a través de mensajes a sus vecinos. Este modelo utiliza las funciones de transformación proporcionadas para implementaciones más directas de cada algoritmo.

### Centrada en la relación

Este enfoque tiene similitudes con el modelo centrado en nodos, pero puede funcionar mejor para el subgrafo y el análisis secuencial.

### Centrado en el grafo

Estos modelos procesan nodos dentro de un subgrafo independientemente de otros subgraphs mientras que la comunicación (mínima) a otros subgraphs ocurre a través de mensajes.

### Centrada en el recorrido

Estos modelos utilizan la acumulación de datos por parte del traverser mientras navega por el gráfico como su medio de cálculo.

### Centrado en el algoritmo

Estos enfoques utilizan varios métodos para optimizar las implementaciones por algoritmo. Este es un híbrido de los modelos anteriores.

### NOTA

[Pregel](#) es un marco de procesamiento paralelo tolerante a fallos, centrado en los nodos, creado por Google para el análisis de grandes gráficos. Pregel se basa en el modelo paralelo síncrono a granel (BSP). BSP simplifica la programación paralela al tener distintas fases de computación y comunicación.

Pregel agrega una abstracción centrada en el nodo en la parte superior de BSP mediante la cual los algoritmos calculan los valores de los mensajes entrantes de los vecinos de cada nodo. Estos cálculos se ejecutan una vez por iteración y pueden actualizar los valores de los nodos y enviar mensajes a otros nodos. Los nodos también pueden combinar mensajes para la transmisión durante la fase de comunicación, lo que reduce de manera útil la cantidad de chatter en la red. El algoritmo se completa cuando no se envían nuevos mensajes o se ha alcanzado un límite establecido.

La mayoría de estos enfoques específicos de gráficos requieren la presencia de todo el gráfico para operaciones eficientes de topología cruzada. Esto se debe a que la separación y distribución de los datos del gráfico conduce a transferencias de datos extensivas y reorganización entre las instancias de los trabajadores. Esto puede ser difícil para los muchos algoritmos que necesitan procesar iterativamente la estructura del gráfico global.

## Plataformas representativas

Para abordar los requisitos del procesamiento de gráficos, han surgido varias plataformas. Tradicionalmente, existía una separación entre los motores de cálculo de gráficos y las bases de datos de gráficos, lo que requería que los usuarios movieran sus datos según sus necesidades de proceso:

### Motores de cálculo gráfico

Estos son motores de solo lectura, no transaccionales, que se centran en la ejecución eficiente de análisis iterativos de gráficos y consultas de todo el gráfico. Los motores de cálculo de gráficos admiten diferentes paradigmas de procesamiento y definición para algoritmos de gráficos, como los centrados en nodos (por ejemplo, Pregel, Gather-Apply-Scatter) o los enfoques basados en MapReduce (por ejemplo, PACT).

Ejemplos de tales motores son Giraph, GraphLab, Graph-Engine y Apache Spark.

### Bases de datos de grafos

Desde un fondo transaccional, estos se enfocan en las escrituras y lecturas rápidas usando consultas más pequeñas que generalmente tocan una pequeña fracción de un gráfico. Sus puntos fuertes están en la robustez operativa y la alta escalabilidad concurrente para muchos usuarios.

## Seleccionando nuestra plataforma

La elección de una plataforma de producción implica muchas consideraciones, como el tipo de análisis a ejecutar, las necesidades de rendimiento, el entorno existente y las preferencias del equipo. Usamos Apache Spark y Neo4j para mostrar los algoritmos gráficos en este libro porque ambos ofrecen ventajas únicas.

Spark es un ejemplo de un motor de cálculo de gráficos de escalamiento horizontal y de nodos. Su popular marco informático y sus bibliotecas admiten una variedad de flujos de trabajo de ciencia de datos. Chispa puede ser la plataforma correcta cuando nuestro:

- Los algoritmos son fundamentalmente paralelizables o particionables.
- Los flujos de trabajo de algoritmos necesitan operaciones "multilingües" en múltiples herramientas e idiomas.
- El análisis se puede ejecutar sin conexión en modo por lotes.
- El análisis gráfico se basa en datos que no se transforman en un formato gráfico.
- El equipo necesita y tiene la experiencia para codificar e implementar sus propios algoritmos.

- El equipo usa algoritmos de gráficos con poca frecuencia.
- El equipo prefiere mantener todos los datos y análisis dentro del ecosistema de Hadoop.

La plataforma gráfica Neo4j es un ejemplo de una base de datos de gráficos muy integrada y un procesamiento centrado en algoritmos, optimizado para gráficos. Es popular para crear aplicaciones basadas en gráficos e incluye una biblioteca de algoritmos de gráficos ajustada para su base de datos de gráficos nativos. Neo4j puede ser la plataforma correcta cuando nuestro:

- Los algoritmos son más iterativos y requieren una buena ubicación de memoria.
- Los algoritmos y los resultados son sensibles al rendimiento.
- El análisis de gráficos se basa en datos de gráficos complejos y / o requiere un recorrido profundo.
- Análisis / resultados están integrados con cargas de trabajo transaccionales.
- Los resultados se utilizan para enriquecer un gráfico existente.
- El equipo necesita integrarse con herramientas de visualización basadas en gráficos.
- El equipo prefiere algoritmos preempaquetados y soportados.

Finalmente, algunas organizaciones utilizan tanto Neo4j como Spark para el procesamiento de gráficos: Spark para el filtrado y preprocesamiento de alto nivel de conjuntos de datos masivos e integración de datos, y Neo4j para un procesamiento más específico e integración con aplicaciones basadas en gráficos.

## Chispa de apache

Apache Spark (a partir de ahora solo Spark) es un motor de análisis para el procesamiento de datos a gran escala. Utiliza una abstracción de tabla llamada DataFrame para representar y procesar datos en filas de columnas nombradas y escritas. La plataforma integra diversos orígenes de datos y admite idiomas como Scala, Python y R. Spark admite varias bibliotecas de análisis, como se muestra en la [Figura 3-1](#). Su sistema basado en memoria funciona utilizando gráficos de cómputo distribuidos de manera eficiente.

GraphFrames es una biblioteca de procesamiento de gráficos para Spark que sucedió a GraphX en 2016, aunque está separada del núcleo Apache Spark. GraphFrames se basa en GraphX, pero utiliza DataFrames como su estructura de datos subyacente. GraphFrames es compatible con los lenguajes de programación Java, Scala y Python. En la primavera de 2019, se aceptó la propuesta "Gráficos de chispas: Gráficos de propiedades, consultas de cifrado y algoritmos" (consulte ["Evolución de gráficos de chispas"](#) ). Esperamos que esto aporte una serie de características gráficas utilizando el marco de DataFrame y el lenguaje de consulta Cypher en el proyecto Spark central. Sin embargo, en este libro, nuestros ejemplos se basarán en la API de Python (PySpark) debido a su popularidad actual entre los científicos de datos de Spark.

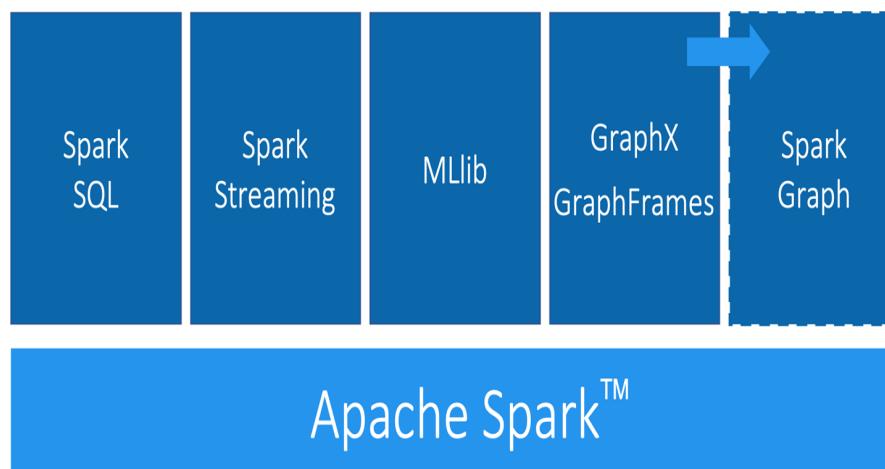


Figura 3-1. Spark es un marco de computación en clúster de código abierto, distribuido y de propósito general. Incluye varios módulos para diferentes cargas de trabajo.

## EVOLUCIÓN DEL GRÁFICO DE CHISPA

[El proyecto Spark Graph](#) es una iniciativa conjunta de los contribuyentes del proyecto Apache en Databricks y Neo4j para brindar soporte para los algoritmos basados en DataFrames, Cypher y DataFrames en el proyecto central Apache Spark como parte de la versión 3.0.

Cypher comenzó como un lenguaje de consulta de gráfico declarativo implementado en Neo4j, pero a través del proyecto openCypher ahora es utilizado por múltiples proveedores de bases de datos y un proyecto de código abierto, [Cypher para Apache Spark \(CAPS\)](#).

En un futuro muy cercano, esperamos utilizar CAPS para cargar y proyectar datos gráficos como una parte integrada de la plataforma Spark. Publicaremos ejemplos de Cypher una vez implementado el proyecto Spark Graph.

Este desarrollo no afecta los algoritmos cubiertos en este libro, pero puede agregar nuevas opciones a cómo se llaman los procedimientos. El modelo de datos subyacente, los conceptos y el cálculo de los algoritmos de gráficos seguirán siendo los mismos.

Los nodos y las relaciones se representan como DataFrames con una ID única para cada nodo y un nodo de origen y destino para cada relación. Podemos ver un ejemplo de un DataFrame de nodos en la [Tabla 3-1](#) y un DataFrame de relaciones en la [Tabla 3-2](#). Un GraphFrame basado en estos DataFrames tendría dos nodos, JFK y SEA , y una relación, de JFK a SEA .

Tabla 3-1. Nodos DataFrame

carné de identidad	ciudad	estado
JFK	Nueva York	Nueva York
MAR	Seattle	Washington

Tabla 3-2. Relaciones DataFrame

src	dst	retrasar	tripId
JFK	MAR	45	1058923

El DataFrame de los nodos debe tener una columna de identificación : el valor de esta columna se utiliza para identificar de forma única cada nodo. Las relaciones DataFrame deben tener columnas src y dst ; los valores en estas columnas describen qué nodos están conectados y deben referirse a las entradas que aparecen en la columna de identificación de los nodos DataFrame.

Los nodos y relaciones DataFrames se pueden cargar utilizando cualquiera de las [fuentes de datos de DataFrame](#) , incluidos Parquet, JSON y CSV. Las consultas se describen utilizando una combinación de la API de PySpark y Spark SQL.

GraphFrames también proporciona a los usuarios [un punto de extensión](#) para implementar algoritmos que no están disponibles de forma inmediata .

## Instalando Chispa

Puedes descargar Spark desde el [sitio web de Apache Spark](#) . Una vez que lo haya descargado, debe instalar las siguientes bibliotecas para ejecutar trabajos de Spark desde Python:

```
pip install pyspark graphframes
```

Luego puede iniciar el pyspark REPL ejecutando el siguiente comando:

```
export SPARK_VERSION = "spark-2.4.0-bin-hadoop2.7" ./${SPARK_VERSION}/bin/pyspark \
--driver-memory 2g \
--executor-memory 6g \
--packages graphframes:graphframes:0.7.0-spark2.4-s_2.11
```

En el momento de escribir la última versión lanzada de Spark es spark-2.4.0-bin-hadoop2.7 , pero eso puede haber cambiado para cuando lea esto. Si es así, asegúrese de cambiar la variable de entorno SPARK\_VERSION adecuadamente.

## NOTA

Aunque los trabajos de Spark deben ejecutarse en un grupo de máquinas, para fines de demostración solo vamos a ejecutar los trabajos en una sola máquina. Puede aprender más sobre cómo ejecutar Spark en entornos de producción en [Spark: The Definitive Guide](#) , por Bill

Chambers y Matei Zaharia (O'Reilly).

Ahora estás listo para aprender a ejecutar algoritmos de gráficos en Spark.

## Plataforma gráfica Neo4j

La plataforma gráfica Neo4j admite el procesamiento transaccional y el procesamiento analítico de datos gráficos. Incluye almacenamiento de gráficos y cómputo con herramientas de análisis y administración de datos. El conjunto de herramientas integradas se encuentra sobre un protocolo, API y lenguaje de consulta comunes (Cypher) para proporcionar un acceso efectivo para diferentes usos, como se muestra en la [Figura 3-2](#).

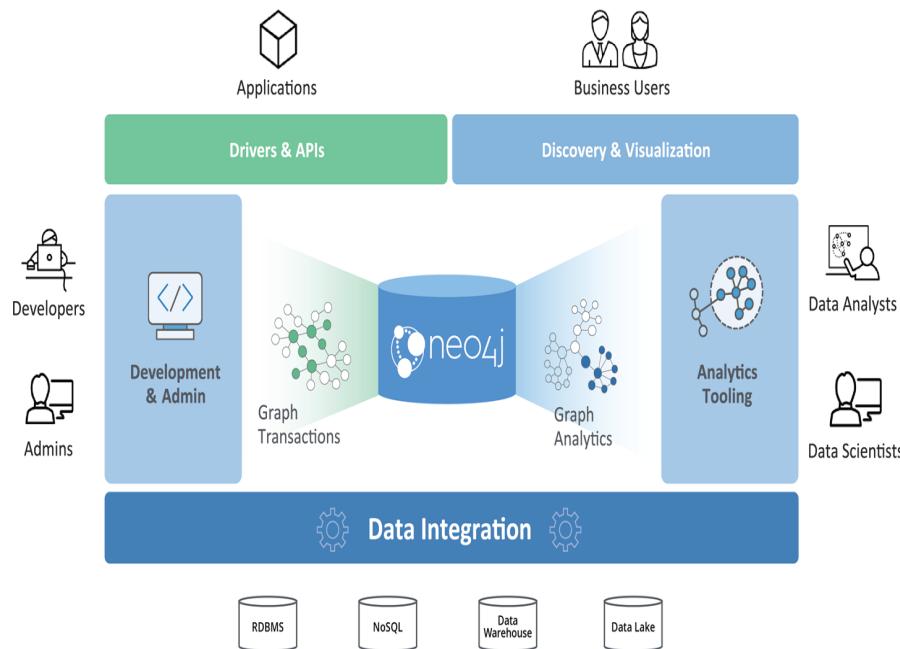


Figura 3-2. La plataforma de gráficos Neo4j se basa en una base de datos de gráficos nativos que admite aplicaciones transaccionales y análisis de gráficos.

En este libro, [usaremos la biblioteca de algoritmos de gráficos Neo4j](#). La biblioteca se instala como un complemento junto a la base de datos y proporciona un conjunto de [procedimientos definidos por el usuario](#) que se pueden ejecutar a través del lenguaje de consulta de Cypher.

La biblioteca de algoritmos de gráficos incluye versiones paralelas de algoritmos que admiten análisis de gráficos y flujos de trabajo de aprendizaje automático. Los algoritmos se ejecutan sobre un marco de computación paralelo basado en tareas y están optimizados para la plataforma Neo4j. Para diferentes tamaños de gráficos hay implementaciones internas que escalan hasta decenas de miles de millones de nodos y relaciones.

Los resultados se pueden transmitir al cliente como un flujo de tuplas y los resultados tabulares se pueden usar como una tabla de manejo para el procesamiento posterior. Los resultados también se pueden escribir opcionalmente en la base de datos de manera eficiente como propiedades de nodo o tipos de relación.

### NOTA

En este libro, también [usaremos la biblioteca Neo4j Awesome Procedures on Cypher \(APOC\)](#). APOC consta de más de 450 procedimientos y funciones para ayudar con tareas comunes como la integración de datos, la conversión de datos y la refactorización de modelos.

## Instalando Neo4j

Neo4j Desktop es una forma conveniente para que los desarrolladores trabajen con las bases de datos locales de Neo4j. Se puede descargar desde el [sitio web de Neo4j](#). Los algoritmos de gráficos y las bibliotecas APOC se pueden instalar como complementos una vez que haya instalado y lanzado el Escritorio Neo4j. En el menú de la izquierda, cree un proyecto y selecciónelo. Luego haga clic en Administrar en la base de datos donde desea instalar los complementos. En la pestaña Complementos, verás opciones para varios complementos. Haga clic en el botón Instalar para algoritmos de gráficos y APOC. Vea las figuras [3-3](#) y [3-4](#).

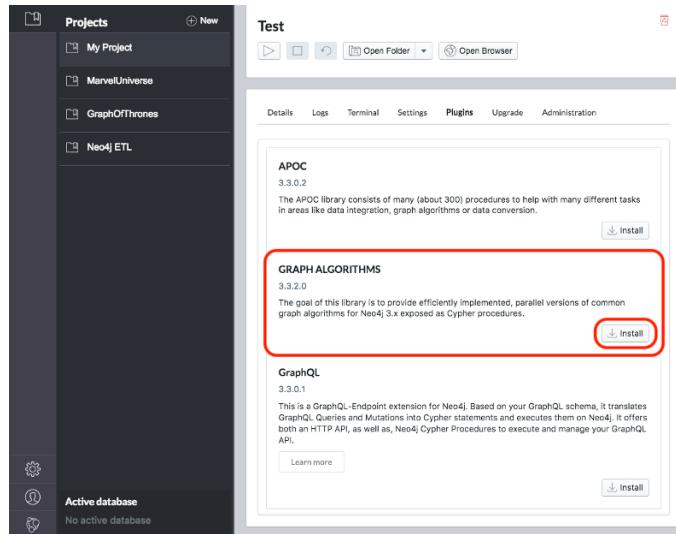


Figura 3-3. Instalando la librería de algoritmos de grafos

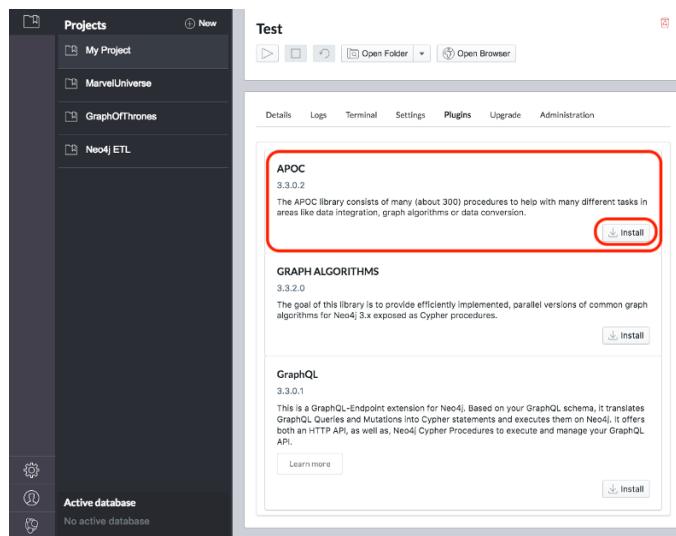


Figura 3-4. Instalación de la librería APOC

Jennifer Reif explica el proceso de instalación con más detalle en la publicación de su blog ["Explorar nuevos mundos: agregar complementos a Neo4j"](#). Ya está listo para aprender a ejecutar algoritmos de gráficos en Neo4j.

## Resumen

En los capítulos anteriores, describimos por qué el análisis de gráficos es importante para estudiar redes del mundo real y analizamos conceptos, análisis y procesamiento de gráficos fundamentales. Esto nos coloca sobre una base sólida para comprender cómo aplicar algoritmos de grafos. En los siguientes capítulos, descubriremos cómo ejecutar algoritmos de gráficos con ejemplos en Spark y Neo4j.

# Capítulo 4. Algoritmos de búsqueda de rutas y búsqueda de gráficos.

---

Gráficos de búsqueda de algoritmos Explorar una gráfica para el descubrimiento general o la búsqueda explícita. Estos algoritmos trazan rutas a través del gráfico, pero no hay ninguna expectativa de que esas rutas sean óptimamente computacionales. Cubriremos la Búsqueda en primer lugar y la Búsqueda en primer lugar porque son fundamentales para recorrer una gráfica y, a menudo, son un primer paso necesario para muchos otros tipos de análisis.

Los algoritmos de búsqueda de rutas se construyen sobre algoritmos de búsqueda de gráficos y exploran rutas entre nodos, comenzando en un nodo y atravesando las relaciones hasta que se ha alcanzado el destino. Estos algoritmos se utilizan para identificar rutas óptimas a través de un gráfico para usos como la planificación logística, llamadas de menor costo o enrutamiento IP y simulación de juegos.

Específicamente, los algoritmos de pathfinding que cubriremos son:

- Ruta más corta, con dos variaciones útiles (A \* y Yen): encontrar la ruta o rutas más cortas entre dos nodos elegidos
- La ruta más corta de todos los pares y la ruta más corta de una sola fuente: para encontrar las rutas más cortas entre todos los pares o desde un nodo elegido a todos los demás
- Árbol de expansión mínimo: para encontrar una estructura de árbol conectada con el menor costo por visitar todos los nodos desde un nodo elegido
- Paseo aleatorio: porque es un paso útil de preprocesamiento / muestreo para flujos de trabajo de aprendizaje automático y otros algoritmos de gráficos

En este capítulo explicaremos cómo funcionan estos algoritmos y mostraremos ejemplos en Spark y Neo4j. En los casos en que un algoritmo solo esté disponible en una plataforma, le proporcionaremos solo un ejemplo o ilustraremos cómo puede personalizar nuestra implementación.

[La Figura 4-1](#) muestra las diferencias clave entre estos tipos de algoritmos, y la [Tabla 4-1](#) es una referencia rápida a lo que cada algoritmo computa con un ejemplo de uso.

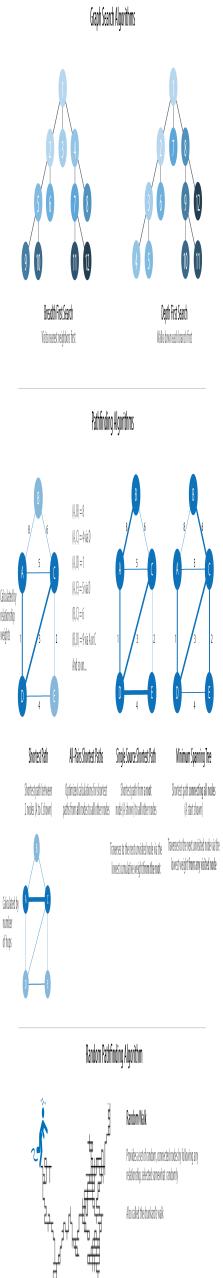


Figura 4-1. Pathfinding y algoritmos de búsqueda.

Tabla 4-1. Visión general de la búsqueda de caminos y algoritmos de búsqueda de gráficos

Tipo de algoritmo	Que hace	Ejemplo de uso	Ejemplo de chispa	Ejemplo neo4j
<u>Amplia primera búsqueda</u>	Atraviesa una estructura de árbol desplegándose para explorar a los vecinos más cercanos y luego a sus vecinos de subnivel	Localización de nodos vecinos en sistemas GPS para identificar lugares de interés cercanos	Sí	No
<u>Primera búsqueda de profundidad</u>	Atraviesa una estructura de árbol explorando en la medida de lo posible cada rama antes de retroceder	Descubrir una ruta de solución óptima en simulaciones de juegos con opciones jerárquicas	No	No
<u>Trayectoria más corta</u> Variaciones: <u>A*</u> , <u>Yen</u>	Calcula la ruta más corta entre un par de nodos	Encontrar direcciones de conducción entre dos ubicaciones	Sí	Sí
<u>El camino más corto de todos los pares</u>	Calcula la ruta más corta entre todos los pares de nodos en el gráfico	Evaluar rutas alternativas alrededor de un atasco de tráfico	Sí	Sí
<u>La ruta más</u>	Calcula la ruta más corta entre un solo nodo raíz y	Enrutamiento de llamadas telefónicas de	Sí	Sí

[corta de una sola fuente](#) todos los demás nodos

menor costo.

<a href="#">Árbol de expansión mínima</a>	Calcula la ruta en una estructura de árbol conectada con el menor costo por visitar todos los nodos	Optimización del enrutamiento conectado, como tendido de cables o recolección de basura	No	Sí
<a href="#">Caminata aleatoria</a>	Devuelve una lista de nodos a lo largo de una ruta de tamaño específico al elegir aleatoriamente las relaciones a atravesar.	Aumento de la formación para el aprendizaje automático o datos para algoritmos de grafos.	No	Sí

Primero, veremos el conjunto de datos de nuestros ejemplos y veremos cómo importar los datos a Apache Spark y Neo4j. Para cada algoritmo, comenzaremos con una breve descripción del algoritmo y cualquier información pertinente sobre cómo funciona. La mayoría de las secciones también incluyen una guía sobre cuándo usar algoritmos relacionados. Finalmente, proporcionamos código de muestra de trabajo utilizando el conjunto de datos de muestra al final de cada sección de algoritmo.

¡Empecemos!

## Datos de ejemplo: el gráfico de transporte

Todos los datos conectados contienen rutas entre nodos, por lo que la búsqueda y la búsqueda de rutas son los puntos de partida para el análisis gráfico. Los conjuntos de datos de transporte ilustran estas relaciones de una manera intuitiva y accesible. Los ejemplos de este capítulo se ejecutan en una [gráfica que contiene un subconjunto de la red de carreteras europea](#). Puede descargar los nodos y los archivos de relaciones desde el [repositorio GitHub del libro](#).

Tabla 4-2. transport-nodes.csv

carné de identidad	latitud	longitud	población
Amsterdam	52.379189	4.899431	821752
Utrecht	52.092876	5.104480	334176
Den Haag	52.078663	4.288788	514861
Immingham	53.61239	-0.22219	9642
Doncaster	53.52285	-1.13116	302400
Hoek van Holland	51.9775	4.13333	9382
Felixstowe	51.96375	1.3511	23689
Ipswich	52.05917	1.15545	133384
Colchester	51.88921	0.90421	104390
Londres	51.509865	-0.118092	8787892
Rotterdam	51.9225	4.47917	623652
Gouda	52.01667	4.70833	70939

Tabla 4-3. transport-relationships.csv

src	dst	relación	costo
Amsterdam	Utrecht	EROAD	46
Amsterdam	Den Haag	EROAD	59
Den Haag	Rotterdam	EROAD	26
Amsterdam	Immingham	EROAD	369
Immingham	Doncaster	EROAD	74
Doncaster	Londres	EROAD	277

Hoek van Holland	Den Haag	EROAD	27
Felixstowe	Hoek van Holland	EROAD	207
Ipswich	Felixstowe	EROAD	22
Colchester	Ipswich	EROAD	32
Londres	Colchester	EROAD	106
Gouda	Rotterdam	EROAD	25
Gouda	Utrecht	EROAD	35
Den Haag	Gouda	EROAD	32
Hoek van Holland	Rotterdam	EROAD	33

[La Figura 4-2](#) muestra el gráfico objetivo que queremos construir.

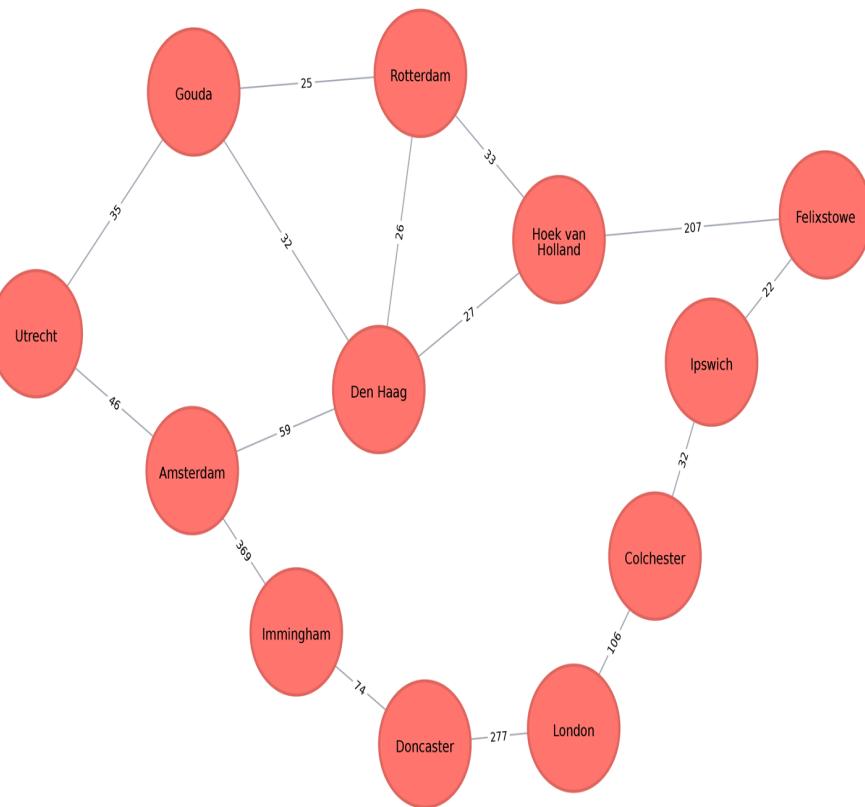


Figura 4-2. La gráfica de transporte

Para simplificar, consideramos que el gráfico de la [Figura 4-2](#) no está dirigido porque la mayoría de los caminos entre ciudades son bidireccionales. Obtendríamos resultados ligeramente diferentes si evaluáramos el gráfico según las indicaciones debido al pequeño número de calles de un solo sentido, pero el enfoque general sigue siendo similar. Sin embargo, tanto Spark como Neo4j operan en gráficos dirigidos. En casos como este, donde queremos trabajar con gráficos no dirigidos (por ejemplo, caminos bidireccionales), hay una manera fácil de lograrlo:

- Para Spark, crearemos dos relaciones para cada fila en transport-relationships.csv: una que va de dst a src y una de src a dst .
- Para Neo4j, crearemos una sola relación y luego ignoraremos la dirección de la relación cuando ejecutemos los algoritmos.

Habiendo entendido esas pequeñas soluciones de modelado, ahora podemos continuar cargando gráficos en Spark y Neo4j desde los archivos CSV de ejemplo.

## Importando los datos en Apache Spark

Comenzando con Spark, primero importaremos los paquetes que necesitamos de Spark y el paquete de GraphFrames:

```
desde pyspark.sql.types import * from graphframes import *
```

La siguiente función crea un GraphFrame a partir de los archivos CSV de ejemplo:

```
def create_transport_graph (): node_fields = [ StructField ( "id" , StringType () , True ), StructField ( "latitude" , FloatType () , True ), StructField ( "longitude" , FloatType () , True ), StructField ( "population" , IntegerType () , True ) ] nodos = chispa . lea . csv ( "data / transport-nodes.csv" , header = True , schema = StructType ( node_fields )) rels = spark . lea . csv ( "datos / transporte-relationships.csv" , encabezado = Verdadero ) reversed_rels = ( REL . withColumn ( "newsrc" , REL . dst ) . withColumnRenamed ( "newDst" , "src" ) . withColumnRenamed ( "newSrc" , "src" ) . drop ( "dst" , "src" ) . Seleccione ( "src" , "dst" , "relación" , "costo" )) relaciones = rels . union ( reversed_rels ) devuelve GraphFrame ( nodos , relaciones )
```

Cargar los nodos es fácil, pero para las relaciones necesitamos hacer un poco de preprocesamiento para poder crear cada relación dos veces.

Ahora llamemos a esa función:

```
g = create_transport_graph ()
```

## Importando los datos en Neo4j

Ahora para Neo4j. Comenzaremos cargando los nodos:

```
CON "https://github.com/neo4j-graph-analytics/book/raw/master/data" AS base WITH base + "transport-nodes.csv" AS uri CSV DE CARGA CON LOS LÍDERES DESDE uri AS fila MERGE (lugar: Coloque { id : row. Id }) SET place.latitude =toFloat (row.latitude), place.longitude =toFloat (row.latitude), place.population =toInteger (row.population)
```

Y ahora las relaciones:

```
CON "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base WITH base + "transport-relationships.csv" AS uri CSV DE CARGA CON LOS JEFATORES DE uri AS fila MATCH (origen : Place { id : row.src }) MATCH (destino: Place { id : row.dst }) MERGE (origen) - [:EROAD {distance: toInteger (row.cost)}] -> (destino)
```

Aunque estamos almacenando relaciones dirigidas, ignoraremos la dirección cuando ejecutemos algoritmos más adelante en el capítulo.

## Amplia primera búsqueda

La búsqueda en primer lugar (BFS) es uno de los algoritmos de recorrido de gráficos fundamentales.

Comienza desde un nodo elegido y explora a todos sus vecinos en un salto antes de visitar a todos los vecinos en dos saltos, y así sucesivamente.

El algoritmo fue publicado por primera vez en 1959 por Edward F. Moore, quien lo utilizó para encontrar el camino más corto para salir de un laberinto. Luego fue desarrollado como un algoritmo de enrutamiento de cables por CY Lee en 1961, como se describe en "[Un algoritmo para conexiones de ruta y sus aplicaciones](#)".

BFS se usa más comúnmente como base para otros algoritmos más orientados a objetivos. Por ejemplo, la [ruta más corta](#), los [componentes conectados](#) y la [centralidad de proximidad](#) utilizan el algoritmo BFS.

También se puede utilizar para encontrar la ruta más corta entre nodos.

[La Figura 4-3](#) muestra el orden en el que visitaríamos los nodos de nuestro gráfico de transporte si estuviéramos realizando una primera búsqueda que comenzó desde la ciudad holandesa, Den Haag (en inglés, La Haya). Los números junto al nombre de la ciudad indican el orden en que se visita cada nodo.

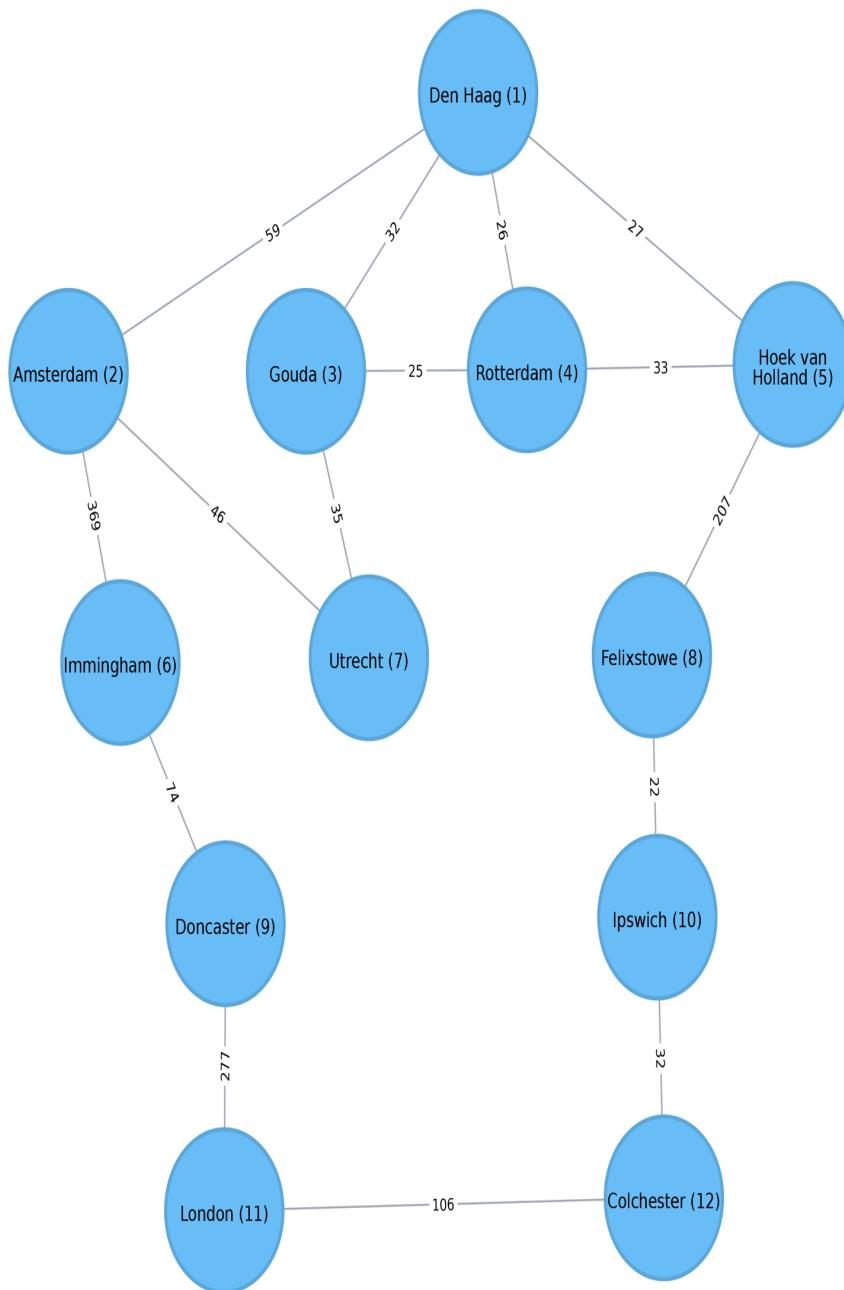


Figura 4-3. Amplia primera búsqueda a partir de Den Haag. Los números de nodo indican el orden atravesado.

Primero visitamos a todos los vecinos directos de Den Haag, antes de visitar a sus vecinos, y a los vecinos de sus vecinos, hasta que nos quedemos sin relaciones para atravesar.

## Amplia primera búsqueda con Apache Spark

La implementación de Spark del algoritmo Breadth First Search encuentra la ruta más corta entre dos nodos por el número de relaciones (es decir, saltos) entre ellos. Puede nombrar explícitamente su nodo de destino o agregar criterios que deben cumplirse.

Por ejemplo, podemos usar la función bfs para encontrar la primera ciudad de tamaño mediano (para los estándares europeos) que tiene una población de entre 100,000 y 300,000 personas. Primero verifiquemos qué lugares tienen una población que coincide con esos criterios:

```
(g . vértices . filtro ( "población > 100000 y población < 300000" ) . sort ( "población" ) . show () )
```

Esta es la salida que veremos:

carné de identidad	latitud	longitude	población
Colchester	51.88921	0.90421	104390
Ipswich	52.05917	1.15545	133384

Solo hay dos lugares que coinciden con nuestros criterios, y esperaríamos llegar a Ipswich primero en base a una amplia búsqueda en primer lugar.

El siguiente código encuentra el camino más corto desde Den Haag a una ciudad de tamaño mediano:

```
from_expr = "id = 'Den Haag'" to_expr = "population > 100000 and population < 300000 and id != 'Den Haag'" result = g . bfs ( from_expr , to_expr )
```

el resultado contiene columnas que describen los nodos y las relaciones entre las dos ciudades. Podemos ejecutar el siguiente código para ver la lista de columnas devueltas:

```
imprimir ( resultado . columnas )
```

Esta es la salida que veremos:

```
[from', 'e0', 'v1', 'e1', 'v2', 'e2', 'to']
```

Las columnas que comienzan con e representan relaciones (bordes) y las columnas que comienzan con v representan nodos (vértices). Solo nos interesan los nodos, así que vamos a filtrar las columnas que comiencen con e desde el DataFrame resultante:

```
columnas = [ columna para la columna en el resultado . columnas si no columna . comienza con ( "e" )] resultado . seleccionar ( columnas ) . mostrar ()
```

Si ejecutamos el código en pyspark veremos esta salida:

desde	v1	v2	a
[Den Haag, 52.078...	[Hoek van Holland ...	[Felixstowe, 51.9...	[Ipswich, 52.0591...

Como era de esperar, el algoritmo bfs devuelve Ipswich! Recuerde que esta función se cumple cuando encuentra la primera coincidencia, y como puede ver en la [Figura 4-3](#), Ipswich se evalúa antes que Colchester.

## Primera búsqueda de profundidad

La búsqueda en profundidad en primer lugar (DFS) es el otro algoritmo de recorrido del gráfico fundamental. Comienza desde un nodo elegido, elige a uno de sus vecinos y luego recorre todo lo que puede en ese camino antes de retroceder.

DFS fue originalmente inventado por el matemático francés Charles Pierre Trémaux como una estrategia para resolver laberintos. Proporciona una herramienta útil para simular posibles caminos para el modelado de escenarios.

[La Figura 4-4](#) muestra el orden en que visitaríamos los nodos de nuestro gráfico de transporte si estuviéramos realizando un DFS que comenzó desde Den Haag.

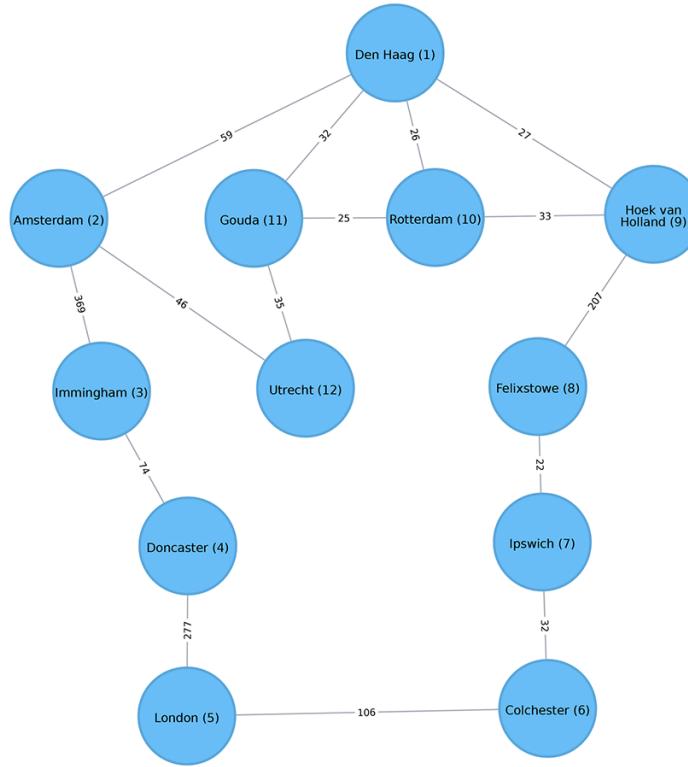


Figura 4-4. Primera búsqueda de profundidad a partir de Den Haag. Los números de nodo indican el orden atravesado.

Observe cuán diferente es el orden del nodo en comparación con BFS. Para este DFS, comenzamos por el recorrido de Den Haag a Ámsterdam, ¡y luego podemos llegar a todos los demás nodos del gráfico sin necesidad de retroceder en absoluto!

Podemos ver cómo los algoritmos de búsqueda sientan las bases para moverse a través de gráficos. Ahora veamos los algoritmos de búsqueda de rutas que encuentran la ruta más barata en términos de número de saltos o peso. Los pesos pueden ser cualquier cosa medida, como el tiempo, la distancia, la capacidad o el costo.

## DOS CAMINOS / CICLOS ESPECIALES

Hay dos caminos especiales en el análisis gráfico que vale la pena señalar. En primer lugar, un camino euleriano es uno en el que cada relación se visita exactamente una vez. En segundo lugar, una ruta hamiltoniana es aquella en la que cada nodo se visita exactamente una vez. Una ruta puede ser a la vez euleriana y hamiltoniana, y si comienza y termina en el mismo nodo, se considera un ciclo o recorrido. Una comparación visual se muestra en la [Figura 4-5](#).

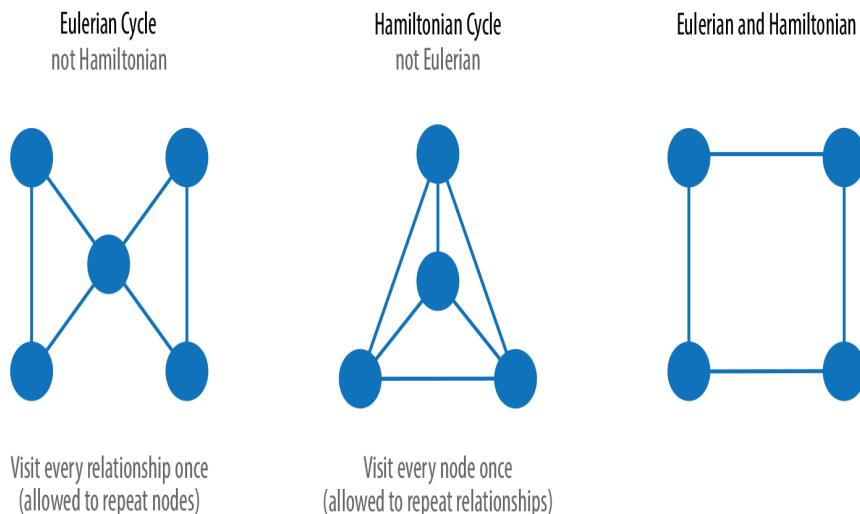


Figura 4-5. Los ciclos eulerianos y hamiltonianos tienen un significado histórico especial.

El problema de puentes de Königsberg del [Capítulo 1](#) buscaba un ciclo de Euler. Es fácil ver cómo se aplica esto a los escenarios de enrutamiento, como dirigir los quitanieves y la entrega de correo. Sin embargo, las rutas Eulerianas también son utilizadas por otros algoritmos para procesar datos en estructuras de árbol y son más sencillas de estudiar matemáticamente que otros ciclos.

El ciclo hamiltoniano es mejor conocido por su relación con el Problema del Vendedor Viajero (TSP), que pregunta: "¿Cuál es la ruta más corta posible para que un vendedor visite cada una de sus ciudades asignadas y regrese a la ciudad de origen?" Aunque parece ser similar a un Eulerian tour, el TSP es computacionalmente más intensivo con alternativas de aproximación. Se utiliza en una amplia variedad de problemas de planificación, logística y optimización..

## Trayectoria más corta

El algoritmo de ruta más corta calcula la ruta más corta (ponderada) entre un par de nodos. Es útil para las interacciones de los usuarios y los flujos de trabajo dinámicos porque funciona en tiempo real.

Pathfinding tiene una historia que se remonta al siglo XIX y se considera un problema de gráficos clásico. Ganó prominencia a principios de la década de 1950 en el contexto de rutas alternativas; es decir, encontrar la segunda ruta más corta si se bloquea la ruta más corta. En 1956, Edsger Dijkstra creó el más conocido de estos algoritmos.

El algoritmo de ruta más corta de Dijkstra funciona al encontrar primero la relación de menor peso desde el nodo de inicio hasta los nodos conectados directamente. Mantiene un registro de esos pesos y se mueve al nodo "más cercano". Luego realiza el mismo cálculo, pero ahora como un total acumulado desde el nodo de inicio. El algoritmo continúa haciendo esto, evaluando una "ola" de pesos acumulativos y siempre eligiendo la ruta acumulada ponderada más baja para avanzar, hasta que llegue al nodo de destino.

### NOTA

Notará en el análisis gráfico el uso de los términos peso , costo , distancia y salto al describir relaciones y caminos. "Peso" es el valor numérico de una propiedad particular de una relación. El "costo" se usa de manera similar, pero lo veremos más a menudo cuando consideremos el peso total de un camino.

La "distancia" se usa a menudo dentro de un algoritmo como el nombre de la propiedad de la relación que indica el costo de atravesar entre un par de nodos. No se requiere que esta sea una medida física real de la distancia. El "salto" se usa comúnmente para expresar el número de relaciones entre dos nodos. Es posible que vea algunos de estos términos combinados, como en "Es una distancia de cinco saltos a Londres" o "Ese es el costo más bajo para la distancia".

## ¿Cuándo debo usar el camino más corto?

Use la ruta más corta para encontrar rutas óptimas entre un par de nodos, según el número de saltos o cualquier valor de relación ponderado. Por ejemplo, puede proporcionar respuestas en tiempo real sobre los grados de separación, la distancia más corta entre puntos o la ruta más económica. También puede usar este algoritmo para explorar simplemente las conexiones entre nodos particulares.

Ejemplos de casos de uso incluyen:

- Encontrar direcciones entre ubicaciones. Las herramientas de mapeo web, como Google Maps, utilizan el algoritmo de ruta más corta, o una variante cercana, para proporcionar instrucciones de manejo.
- Encontrar los grados de separación entre personas en las redes sociales. Por ejemplo, cuando ve el perfil de alguien en LinkedIn, indicará cuántas personas lo separan en el gráfico, así como una lista de sus conexiones mutuas.
- Encontrar el número de grados de separación entre un actor y Kevin Bacon en función de las películas en las que han aparecido (el Número de tocino ). Un ejemplo de esto se puede ver en el [sitio web de Oracle of Bacon](#) . [El Proyecto Número Erdős](#) proporciona un análisis gráfico similar basado en la colaboración con Paul Erdős, uno de los matemáticos más prolíficos del siglo XX.

### PROPINA

El algoritmo de Dijkstra no soporta pesos negativos. El algoritmo asume que agregar una relación a una ruta nunca puede hacer que una ruta sea más corta, una invariante que se violaría con pesos negativos.

## El camino más corto con Neo4j

La biblioteca de algoritmos de gráficos Neo4j tiene un procedimiento incorporado que podemos usar para calcular tanto las rutas más cortas no ponderadas como las ponderadas. Primero aprendamos a calcular los caminos más cortos no ponderados.

### PROPIA

Todos los algoritmos de la ruta más corta de Neo4j asumen que el gráfico subyacente no está dirigido. Puede anular esto pasando la dirección del parámetro : "SALIENTES" o la dirección: "ENTRANTES" .

Para que el algoritmo de la ruta más corta de Neo4j ignore los pesos, debemos pasar nulo como tercer parámetro al procedimiento, lo que indica que no queremos considerar una propiedad de peso al ejecutar el algoritmo. El algoritmo asumirá entonces un peso predeterminado de 1.0 para cada relación:

```
MATCH (fuente: Place { id : "Amsterdam" }), (destino: Place { id : "London" }) CALL algo.shortestPath.stream (source, destination, null ) nodeId YIELD costo RETURN algo.getNodeById (nodeId) . Identificación del AS lugar, el costo
```

Esta consulta devuelve el siguiente resultado:

lugar	costo
Amsterdam	0.0
Immingham	1.0
Doncaster	2.0
Londres	3.0

Aquí el costo es el total acumulado para las relaciones (o saltos). Este es el mismo camino que vemos usando la Búsqueda de amplitud en primer lugar en Spark.

Incluso podríamos calcular la distancia total de seguir este camino escribiendo un poco de Cypher de posprocesamiento. El siguiente procedimiento calcula la ruta no ponderada más corta y luego determina cuál sería el costo real de esa ruta:

```
MATCH (fuente: Place { id : "Amsterdam" }), (destino: Place { id : "London" }) CALL algo.shortestPath.stream (source, destination, null ) YIELD nodeId, costo WITH collect (algo.getNodeById ( nodeID )) AS path UNWIND intervalo (0, tamaño (path) -1) AS index CON path path [index] AS current, path [index + 1] AS next WITH current, next, [(current) - [r: EROAD] - (siguiente) | r.distance] [0] COMO distancia CON recogida      ({current: current, next: next, distance: distance}) AS detiene el rango UNWIND (0, tamaño (se detiene) -1) AS index WITH se detiene [index] AS location, se detiene, index RETURN location.current. id AS place, reduce (acc = 0.0, distancia en [stop en stops [0..index] | stop.distance] | acc + distancia) AS costo
```

Si el código anterior se siente un poco difícil de manejar, observe que la parte difícil es averiguar cómo masajear los datos para incluir el costo en todo el viaje. Esto es útil para tener en cuenta cuando necesitamos el costo acumulado de la ruta.

La consulta devuelve el siguiente resultado:

lugar	costo
Amsterdam	0.0
Immingham	369.0
Doncaster	443.0
Londres	720.0

[La Figura 4-6](#) muestra el camino más corto no ponderado de Amsterdam a Londres, guiándonos a través del menor número de ciudades. Tiene un coste total de 720 km.

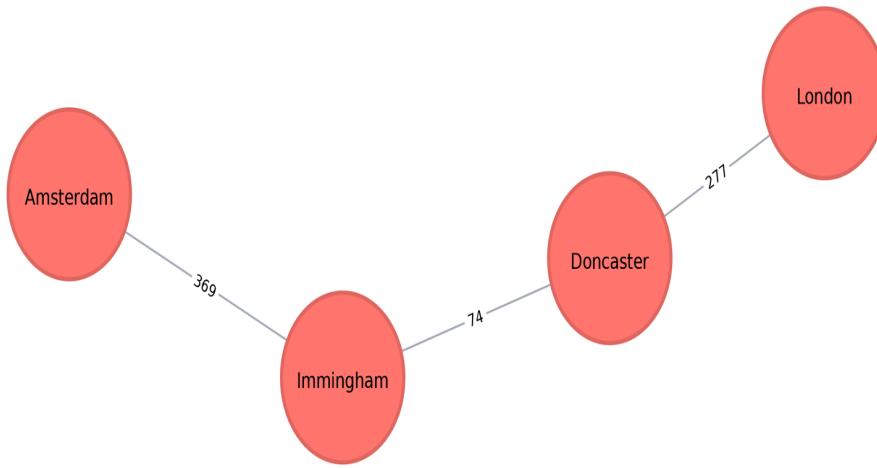


Figura 4-6. El camino más corto no ponderado entre Amsterdam y Londres

La elección de una ruta con el menor número de nodos visitados puede ser muy útil en situaciones como los sistemas de metro, donde son muy deseables hacer menos paradas. Sin embargo, en un escenario de conducción, probablemente estemos más interesados en el costo total utilizando la ruta ponderada más corta.

## Ruta más corta (ponderada) con Neo4j

Podemos ejecutar el algoritmo de ruta más corta ponderada para encontrar la ruta más corta entre Amsterdam y Londres de la siguiente manera:

```

MATCH (fuente: Place { id : "Amsterdam" }), (destination: Place { id : "London" }) CALL algo.shortestPath.stream (source,
destination, "distance" ) YIELD nodeId, costo RETURN algo.getNodeById ( nodeId). Identificación del AS lugar, el costo
  
```

Los parámetros pasados a este algoritmo son:

**fuente**

El nodo donde comienza nuestra búsqueda de ruta más corta.

**destino**

El nodo donde termina nuestro camino más corto.

**distancia**

El nombre de la propiedad de relación que indica el costo de atravesar entre un par de nodos

El costo es el número de kilómetros entre dos lugares. La consulta devuelve el siguiente resultado:

lugar	costo
Amsterdam	0.0
Den Haag	59.0
Hoek van Holland	86.0
Felixstowe	293.0
Ipswich	315.0
Colchester	347.0
Londres	453.0

¡La ruta más rápida nos lleva por Den Haag, Hoek van Holland, Felixstowe, Ipswich y Colchester! El costo que se muestra es el total acumulado a medida que avanzamos a través de las ciudades. Primero vamos de Amsterdam a Den Haag, a un costo de 59. Luego, vamos de Den Haag a Hoek van Holland, a un costo acumulado de 86, y así sucesivamente. Finalmente, llegamos a Londres, desde Colchester, por un costo total de 453 km.

Recuerde que la ruta más corta no ponderada tuvo un costo total de 720 km, por lo que hemos podido ahorrar 267 km teniendo en cuenta los pesos al calcular la ruta más corta.

## Ruta más corta (ponderada) con Apache Spark

En la sección de [Búsqueda de amplitud con Apache Spark](#), aprendimos a encontrar la ruta más corta entre dos nodos. El camino más corto se basó en saltos y, por lo tanto, no es el mismo que el camino ponderado más corto, lo que nos diría la distancia total más corta entre ciudades.

Si queremos encontrar la ruta ponderada más corta (en este caso, la distancia) necesitamos usar la propiedad de costo, que se usa para varios tipos de ponderación. Esta opción no está disponible fuera de la caja con GraphFrames, por lo que necesitamos escribir nuestra propia versión de Weighted Shortest Path utilizando su [marco de agregación de mensajes](#). La mayoría de nuestros ejemplos de algoritmos para Spark utilizan el proceso más simple para llamar algoritmos de la biblioteca, pero tenemos la opción de escribir nuestras propias funciones. Más información sobre aggregateMessages se puede encontrar en el “[mensaje que pasa a través de AggregateMessages](#)” sección de la guía del usuario GraphFrames.

### PROPIA

Cuando esté disponible, recomendamos aprovechar bibliotecas probadas y preexistentes. Escribir nuestras propias funciones, especialmente para algoritmos más complicados, requiere una comprensión más profunda de nuestros datos y cálculos.

El siguiente ejemplo debe tratarse como una implementación de referencia y debería optimizarse antes de ejecutarse en un conjunto de datos más grande. Aquellos que no estén interesados en escribir sus propias funciones pueden omitir este ejemplo.

Antes de crear nuestra función, importaremos algunas bibliotecas que usaremos:

```
desde graphframes.lib importa AggregateMessages como AM desde pyspark.sql las funciones de importación como F
```

El módulo Aggregate\_Messages es parte de la biblioteca de GraphFrames y contiene algunas funciones de ayuda útiles.

Ahora vamos a escribir nuestra función. Primero creamos una función definida por el usuario que usaremos para construir las rutas entre nuestro origen y destino:

```
add_path_udf = F.udf(ruta lambda , id : ruta + [ id ], ArrayType ( StringType () ))
```

Y ahora para la función principal, que calcula la ruta más corta a partir de un origen y regresa tan pronto como el destino ha sido visitado:

```
def shortest_path ( g , origen , destino , nombre_columna = "costo" ): si g . vértices . filtro ( g . vértices . id == destino ) . count () == 0 :  
    return ( spark . createDataFrame ( sc . emptyRDD (), g . vertices . schema ) . withColumn ( "camino" , F . array () ) )  
    vértices = ( g . vértices . withColumn ( "visitada" , F . iluminado ( Falso )) . withColumn ( "distancia" , F . , cuando ( g . vértices [ "id" ] == origen , 0 ) . de lo contrario ( float ( " inf" ))) . withColumn ( "camino" , F . array () ) )  
    cached_vertices = AM . getCachedDataFrame ( vértices ) g2 = GraphFrame ( cached_vertices , g . bordes ) mientras g2 . vértices . filtro ( 'visitado == Falso' ) . primero () : current_node_id = g2 . vértices . filtro ( 'visitado == Falso' ) . ordenar ( "distancia" ) . primero () . id msg_distance = AM . borde [ column_name ] + AM . src [ 'distancia' ]  
    msg_path = add_path_udf ( AM . src [ "camino" ], AM . src [ "id" ]) msg_for_dst = F . cuando ( AM . src [ 'id' ] == current_node_id , F . struct ( msg_distance , msg_path )) new_distances = g2 . aggregateMessages ( F . min ( AM . MSG ) . alias ( "aggMess" ), sendToDst = msg_for_dst ) new_visited_col = F . cuando ( g2 . vértices . visitado | ( g2 . vértices . id == current_node_id ), True ) . de lo contrario ( Falso ) new_distance_col = F . cuando ( new_distances [ "aggMess" ] . isNotNull () & ( new_distances . aggMess [ "col1" ] < g2 . vertices . distance ), new_distances . aggMess [ "col1" ]) . de lo contrario ( g2 . vértices . distancia ) new_path_col = F . cuando ( new_distances [ "aggMess" ] . isNotNull () & ( new_distances . aggMess [ "col1" ] < g2 . vertices . distance ), new_distances . aggMess [ "col2" ]) . cast ( "array <string>" ) . De lo contrario ( g2 . vértices . ruta ) new_vertices = ( g2 . vértices . join ( new_distances , on = "id" , how = "left_outer" ) . drop ( new_distances [ "id" ]) . withColumn ( "visited" , new_visited_col ) . withColumn ( "newDistance" , new_distance_col ) . withColumn ( "newPath" , new_path_col ) . drop ( "aggMess" , "distance" , "path" ) . withColumnRenamed ( 'newDistance' , 'distance' ) . withColumnRenamed ( 'newPath' , 'path' ))  
    cached_new_vertices = AM . getCachedDataFrame ( new_vertices ) g2 = GraphFrame ( cached_new_vertices , g2 . bordes ) si g2 . vértices . filtrar ( g2 . vértices . id == destino ) . primero () . visitó : retorno ( g2 . vértices . filtro ( g2 . vértices . id == destino ) . withColumn ( "newPath" , add_path_udf ( "path" , "id" )) . drop ( "visited" , "path" ) . withColumnRenamed ( "Newpath" , "camino" )) de retorno ( chispa . CreateDataFrame ( sc . EmptyRDD (), g . Vértices . Esquema ) . WithColumn ( "camino" , F . Array ()) )
```

### ADVERTENCIA

Si almacenamos referencias a cualquier DataFrame en nuestras funciones, necesitamos almacenarlas en caché usando la función AM.getCacheDataFrame o encontraremos una pérdida de memoria durante la ejecución. En el shortest\_path función se utiliza esta función para almacenar en caché los vértices y new\_vertices tramas de datos.

Si quisieramos encontrar el camino más corto entre Amsterdam y Colchester, podríamos llamar a esa función así:

```
resultado = shortest_path ( g , "Amsterdam" , "Colchester" , "costo" ) resultado . seleccione ( "id" , "distancia" , "ruta" ) . show ( truncado = falso )
```

que devolvería el siguiente resultado:

#### carné de identidad distancia camino

Colchester	347.0	[Ámsterdam, Den Haag, Hoek van Holland, Felixstowe, Ipswich, Colchester]
------------	-------	--

La distancia total del camino más corto entre Ámsterdam y Colchester es de 347 km y nos lleva por Den Haag, Hoek van Holland, Felixstowe e Ipswich. Por el contrario, el camino más corto en términos de número de relaciones entre las ubicaciones, que trabajamos con el [algoritmo de búsqueda de Breadth First](#) (consulte la [Figura 4.4](#)), nos llevaría a través de Immingham, Doncaster y Londres.

## Variación del camino más corto: A \*

El algoritmo A \* Shortest Path mejora en Dijkstra al encontrar las rutas más cortas más rápidamente. Para ello, permite la inclusión de información adicional que el algoritmo puede utilizar, como parte de una función heurística, al determinar qué rutas explorar a continuación.

El algoritmo fue inventado por Peter Hart, Nils Nilsson y Bertram Raphael y se describió en su artículo de 1968 ["Una base formal para la determinación heurística de las rutas de costo mínimo"](#).

El algoritmo A \* opera determinando cuál de sus caminos parciales se expandirá en cada iteración de su bucle principal. Lo hace basándose en una estimación del costo (heurístico) que aún queda para alcanzar el nodo objetivo.

### ADVERTENCIA

Sea considerado en la heurística empleada para estimar los costos de la ruta. La subestimación de los costos de la ruta puede incluir innecesariamente algunas rutas que podrían haberse eliminado, pero los resultados seguirán siendo precisos. Sin embargo, si la heurística sobreestima los costos de la ruta, puede omitir las rutas más cortas reales (estimadas incorrectamente como más largas) que deberían haberse evaluado, lo que puede dar lugar a resultados inexactos.

A \* selecciona la ruta que minimiza la siguiente función:

$$f(n) = g(n) + h(n)$$

dónde:

- G (n) es el costo de la ruta desde el punto de inicio hasta el nodo n .
- H (n) es el costo estimado de la ruta desde el nodo n al nodo de destino, según lo calculado por una heurística.

### NOTA

En la implementación de Neo4j, la distancia geoespacial se utiliza como heurística. En nuestro ejemplo de conjunto de datos de transporte, usamos la latitud y longitud de cada ubicación como parte de la función heurística.

### A \* con Neo4j

La siguiente consulta ejecuta el algoritmo A \* para encontrar la ruta más corta entre Den Haag y Londres:

```
MATCH (fuente: Place { id : "Den Haag" }), (destination: Place { id : "London" }) CALL algo.shortestPath.astar.stream (source, destination, "distance" , "latitude" , "longitude" ) YIELD nodeId, el costo RETURN algo.getNodeById (nodeId). Identificación del AS lugar, el costo
```

Los parámetros pasados a este algoritmo son:

fuente

El nodo donde comienza nuestra búsqueda de ruta más corta.

destino

El nodo donde termina nuestra búsqueda de ruta más corta.

distancia

El nombre de la propiedad de relación que indica el costo de atravesar entre un par de nodos. El costo es el número de kilómetros entre dos lugares.

latitud

El nombre de la propiedad de nodo utilizada para representar la latitud de cada nodo como parte del cálculo heurístico geoespacial.

longitud

El nombre de la propiedad de nodo utilizada para representar la longitud de cada nodo como parte del cálculo heurístico geoespacial.

Ejecutar este procedimiento da el siguiente resultado:

<b>lugar</b>	<b>costo</b>
Den Haag	0.0
Hoek van Holland	27.0
Felixstowe	234.0
Ipswich	256.0
Colchester	288.0
Londres	394.0

Obtendríamos el mismo resultado utilizando el algoritmo de ruta más corta, pero en conjuntos de datos más complejos, el algoritmo A \* será más rápido ya que evalúa menos rutas.

## Variación del camino más corto: los k-caminos más cortos de Yen

El algoritmo K- Shortest Paths de Yen es similar al algoritmo de ruta más corta, pero en lugar de encontrar la ruta más corta entre dos pares de nodos, también calcula la segunda ruta más corta, la tercera ruta más corta, y así sucesivamente hasta k -1 desviaciones de Los caminos más cortos.

Jin Y. Yen inventó el algoritmo en 1971 y lo describió en "[Cómo encontrar las K rutas sin lazo más cortas en una red](#)". Este algoritmo es útil para obtener rutas alternativas cuando encontrar la ruta más corta absoluta no es nuestro único objetivo. ¡Puede ser particularmente útil cuando necesitamos más de un plan de respaldo!

### Yen con Neo4j

La siguiente consulta ejecuta el algoritmo de Yen para encontrar las rutas más cortas entre Gouda y Felixstowe:

```
MATCH ( inicio : Coloque { id : "Gouda" }), (final: Coloque { id : "Felixstowe" }) LLAME algo.kShortestPaths.stream ( inicio , fin, 5, "distancia" ) índice de RENDIMIENTO, nodeIds, ruta, cuesta el índice de RETORNO , [ nodo en algo.getNodesById (nodeIds [1 .. 1]) | nodo . id ] AS via, reduce (acc = 0.0, costo en costos | acc + cost) AS totalCost
```

Los parámetros pasados a este algoritmo son:

comienzo

El nodo donde comienza nuestra búsqueda de ruta más corta.

fin

El nodo donde termina nuestra búsqueda de ruta más corta.

5

El número máximo de caminos más cortos para encontrar.

distancia

El nombre de la propiedad de relación que indica el costo de atravesar entre un par de nodos. El costo es el número de kilómetros entre dos lugares.

Después de recuperar las rutas más cortas, buscamos el nodo asociado para cada ID de nodo y luego filtramos los nodos de inicio y final de la colección.

Ejecutar este procedimiento da el siguiente resultado:

índice vía		coste total
0	[Rotterdam, Hoek van Holland]	265.0
1	[Den Haag, Hoek van Holland]	266.0
2	[Rotterdam, Den Haag, Hoek van Holland]	285.0
3	[Den Haag, Rotterdam, Hoek van Holland]	298.0
4	[Utrecht, Amsterdam, Den Haag, Hoek van Holland]	374.0

La Figura 4-7 muestra el camino más corto entre Gouda y Felixstowe.

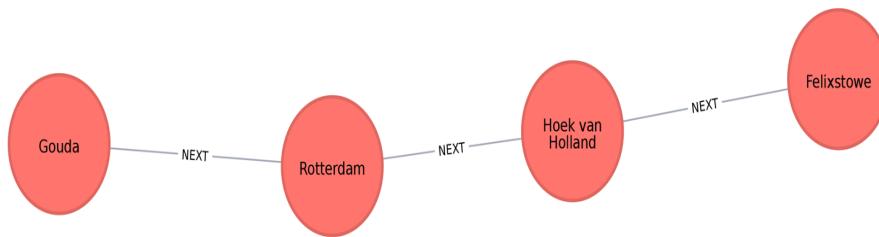


Figura 4-7. El camino más corto entre Gouda y Felixstowe

La ruta más corta en la Figura 4-7 es interesante en comparación con los resultados ordenados por costo total. Ilustra que a veces es posible que desee considerar varias rutas más cortas u otros parámetros. En este ejemplo, la segunda ruta más corta es solo 1 km más larga que la más corta. Si preferimos el paisaje, podríamos elegir la ruta un poco más larga.

## El camino más corto de todos los pares

El algoritmo de ruta más corta de todos los pares (APSP) calcula la ruta más corta (ponderada) entre todos los pares de nodos. Es más eficiente que ejecutar el algoritmo de ruta más corta de una sola fuente para cada par de nodos en el gráfico.

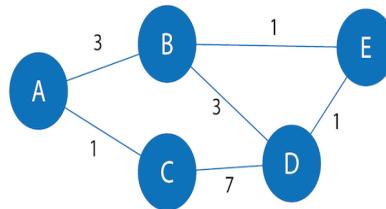
APSP optimiza las operaciones mediante el seguimiento de las distancias calculadas hasta el momento y la ejecución en nodos en paralelo. Esas distancias conocidas se pueden reutilizar cuando se calcula la ruta más corta a un nodo invisible. Puede seguir el ejemplo de la siguiente sección para comprender mejor cómo funciona el algoritmo.

### NOTA

Es posible que algunos pares de nodos no sean accesibles entre sí, lo que significa que no hay una ruta más corta entre estos nodos. El algoritmo no devuelve distancias para estos pares de nodos.

## Una mirada más cercana a todos los pares camino más corto

El cálculo para APSP es más fácil de entender cuando se sigue una secuencia de operaciones. El diagrama de la Figura 4-8 muestra los pasos para el nodo A.



			Each Step Keeps or Updates to the Lowest Value Calculated so Far Only steps for node A to all nodes shown				
All nodes start with a $\infty$ distance and then the start node is set to a 0 distance			1 <sup>st</sup> from A	2 <sup>nd</sup> from A to C to Next	3 <sup>rd</sup> from A to B to Next	4 <sup>th</sup> from A to E to Next	5 <sup>th</sup> from A to D to Next
A	$\infty$	0	0	0	0	0	0
B	$\infty$	$\infty$	3	3	3	3	3
C	$\infty$	$\infty$	1	1	1	1	1
D	$\infty$	$\infty$	$\infty$	8	6	5	5
E	$\infty$	$\infty$	$\infty$	$\infty$	4	4	4

Figura 4-8. Los pasos para calcular la ruta más corta desde el nodo A a todos los demás nodos, con las actualizaciones sombreadas.

Inicialmente, el algoritmo asume una distancia infinita a todos los nodos. Cuando se selecciona un nodo de inicio, la distancia a ese nodo se establece en 0. El cálculo continúa de la siguiente manera:

1. Desde el nodo de inicio A, evaluamos el costo de mudarnos a los nodos a los que podemos alcanzar y actualizar esos valores. Buscando el valor más pequeño, tenemos una opción de B (costo de 3) o C (costo de 1). C se selecciona para la siguiente fase de recorrido.
2. Ahora, desde el nodo C, el algoritmo actualiza las distancias acumuladas desde A hasta los nodos a los que se puede llegar directamente desde C. Los valores solo se actualizan cuando se encuentra un costo menor:

$$A = 0, B = 3, C = 1, D = 8, E =$$

3. Luego se selecciona B como el siguiente nodo más cercano que aún no se ha visitado. Tiene relaciones con los nodos A, D y E. El algoritmo calcula la distancia a esos nodos sumando la distancia de A a B con la distancia de B a cada uno de esos nodos. Tenga en cuenta que el costo más bajo desde el nodo de inicio A hasta el nodo actual siempre se conserva como un costo irrecuperable. Los resultados del cálculo de la distancia (d):

$$d(A, A) = d(A, B) + d(B, A) = 3 + 3 = 6 \quad d(A, D) = d(A, B) + d(B, D) = 3 + 3 = 6 \quad d(A, E) = d(A, B) + d(B, E) = 3 + 1 = 4$$

- En este paso, la distancia del nodo A al B y de regreso a A, mostrada como  $d(A, A) = 6$ , es mayor que la distancia más corta ya calculada (0), por lo que su valor no se actualiza.
  - Las distancias para los nodos D (6) y E (4) son menores que las distancias calculadas previamente, por lo que sus valores se actualizan.
4. E se selecciona a continuación. Solo el total acumulado para alcanzar D (5) ahora es más bajo y, por lo tanto, es el único actualizado.
  5. Cuando D finalmente se evalúa, no hay nuevos pesos mínimos de ruta; nada se actualiza, y el algoritmo termina.

## PROPIÑA

A pesar de que el algoritmo de ruta más corta de todos los pares está optimizado para ejecutar cálculos en paralelo para cada nodo, esto puede sumar para un gráfico muy grande. Considere usar un subgrafo si solo necesita evaluar rutas entre una subcategoría de nodos.

## ¿Cuándo debo usar el camino más corto de todos los pares?

La ruta más corta de todos los pares se usa comúnmente para entender el enrutamiento alternativo cuando la ruta más corta está bloqueada o se vuelve subóptima. Por ejemplo, este algoritmo se usa en la planificación de rutas lógicas para garantizar las mejores rutas múltiples para el enrutamiento de diversidad. Use la ruta más corta de todos los pares cuando necesite considerar todas las rutas posibles entre todos o la mayoría de sus nodos.

Ejemplos de casos de uso incluyen:

- Optimizar la ubicación de las instalaciones urbanas y la distribución de mercancías. Un ejemplo de esto es determinar la carga de tráfico esperada en diferentes segmentos de una red de transporte. Para obtener más información, consulte el libro de RC Larson y AR Odoni, *Urban Operations Research* (Prentice-Hall).
- Encontrar una red con un ancho de banda máximo y una latencia mínima como parte de un algoritmo de diseño de centro de datos. Hay más detalles sobre este enfoque en el documento [“REWIRE: Un marco basado en la optimización para el diseño de redes de centros de datos”](#), por AR Curtis et al.

## El camino más corto de todos los pares con Apache Spark

De chispa shortestPaths función está diseñada para encontrar las rutas más cortas desde todos los nodos a un conjunto de nodos llamados puntos de referencia . Si quisiéramos encontrar el camino más corto desde cada ubicación a Colchester, Immingham y Hoek van Holland, escribiríamos la siguiente consulta:

```
resultado = g . shortestPaths ([ "Colchester" , "Immingham" , "Hoek van Holland" ]) resultado . ordenar ([ "id" ]) . seleccione ( "id" , "distancias" ) . show ( truncado = falso )
```

Si ejecutamos ese código en pyspark veremos esta salida:

### carné de identidad distancias

Amsterdam	[Immingham → 1, Hoek van Holland → 2, Colchester → 4]
Colchester	[Colchester → 0, Hoek van Holland → 3, Immingham → 3]
Den Haag	[Hoek van Holland → 1, Immingham → 2, Colchester → 4]
Doncaster	[Immingham → 1, Colchester → 2, Hoek van Holland → 4]
Felixstowe	[Hoek van Holland → 1, Colchester → 2, Immingham → 4]
Gouda	[Hoek van Holland → 2, Immingham → 3, Colchester → 5]
Hoek van Holland	[Hoek van Holland → 0, Immingham → 3, Colchester → 3]
Immingham	[Immingham → 0, Colchester → 3, Hoek van Holland → 3]
Ipswich	[Colchester → 1, Hoek van Holland → 2, Immingham → 4]
Londres	[Colchester → 1, Immingham → 2, Hoek van Holland → 4]
Rotterdam	[Hoek van Holland → 1, Immingham → 3, Colchester → 4]
Utrecht	[Immingham → 2, Hoek van Holland → 3, Colchester → 5]

El número al lado de cada ubicación en la columna de distancias es el número de relaciones (caminos) entre ciudades que necesitamos recorrer para llegar desde el nodo de origen. En nuestro ejemplo, Colchester es una de nuestras ciudades de destino y puede ver que tiene 0 nodos que recorrer para llegar a sí mismo, pero 3 saltos para hacer desde Immingham y Hoek van Holland. Si estuviéramos planeando un viaje, podríamos usar esta información para ayudarnos a maximizar nuestro tiempo en nuestros destinos elegidos.

## El camino más corto de todos los pares con Neo4j

Neo4j tiene una implementación paralela del algoritmo de ruta más corta de todos los pares, que devuelve la distancia entre cada par de nodos.

El primer parámetro de este procedimiento es la propiedad que se utiliza para calcular la ruta ponderada más corta. Si establecemos esto en nulo , el algoritmo calculará las rutas más cortas no ponderadas entre todos los pares de nodos.

La siguiente consulta hace esto:

```
CALL algo.allShortestPaths.stream ( null ) YIELD sourceNodeId, targetNodeId, distancia
WHERE sourceNodeId < targetNodeId
RETURN algo.getNodeById (sourceNodeId). id AS source, algo.getNodeById (targetNodeId). id AS target, distance
ORDER BY distancia DESC LIMIT 10
```

Este algoritmo devuelve la ruta más corta entre cada par de nodos dos veces, una vez con cada uno de los nodos como el nodo de origen. Esto sería útil si estuviera evaluando un gráfico dirigido de calles de un solo sentido. Sin embargo, no necesitamos ver cada ruta dos veces, así que filtramos los resultados para mantener solo una de ellas utilizando el predicado sourceNodeId < targetNodeId .

La consulta devuelve los siguientes resultados:

fuente	objetivo	distancia
Colchester	Utrecht	5.0
Londres	Rotterdam	5.0
Londres	Gouda	5.0
Ipswich	Utrecht	5.0
Colchester	Gouda	5.0
Colchester	Den Haag	4.0
Londres	Utrecht	4.0
Londres	Den Haag	4.0
Colchester	Amsterdam	4.0
Ipswich	Gouda	4.0

Esta salida muestra los 10 pares de ubicaciones que tienen más relaciones entre ellos porque solicitamos resultados en orden descendente ( DESC ).

Si queremos calcular las rutas ponderadas más cortas, en lugar de pasar el valor nulo como primer parámetro, podemos pasar el nombre de la propiedad que contiene el costo que se utilizará en el cálculo de la ruta más corta. Esta propiedad se evaluará para determinar la ruta ponderada más corta entre cada par de nodos.

La siguiente consulta hace esto:

```
CALL algo.allShortestPaths.stream ( "distance" ) YIELD sourceNodeId, targetNodeId, distance
WHERE sourceNodeId < targetNodeId
RETURN algo.getNodeById (sourceNodeId). id AS source, algo.getNodeById (targetNodeId). id AS target, distance
ORDER BY distancia DESC LIMIT 10
```

La consulta devuelve el siguiente resultado:

fuente	objetivo	distancia
Doncaster	Hoek van Holland	529.0
Rotterdam	Doncaster	528.0
Gouda	Doncaster	524.0
Felixstowe	Immingham	511.0
Den Haag	Doncaster	502.0
Ipswich	Immingham	489.0

Utrecht	Doncaster	489.0
Londres	Utrecht	460.0
Colchester	Immingham	457.0
Immingham	Hoek van Holland	455.0

Ahora estamos viendo los 10 pares de ubicaciones más alejadas entre sí en términos de la distancia total entre ellos. Tenga en cuenta que Doncaster aparece con frecuencia junto con varias ciudades en los Países Bajos. Parece que sería un viaje largo si quisieramos hacer un viaje por carretera entre esas áreas.

## La ruta más corta de una sola fuente

El algoritmo de ruta más corta de una sola fuente (SSSP), que se destacó casi al mismo tiempo que el algoritmo de la ruta más corta de Dijkstra, actúa como una implementación para ambos problemas.

El algoritmo SSSP calcula la ruta más corta (ponderada) desde un nodo raíz a todos los demás nodos en el gráfico, como se muestra en la [Figura 4-9](#).

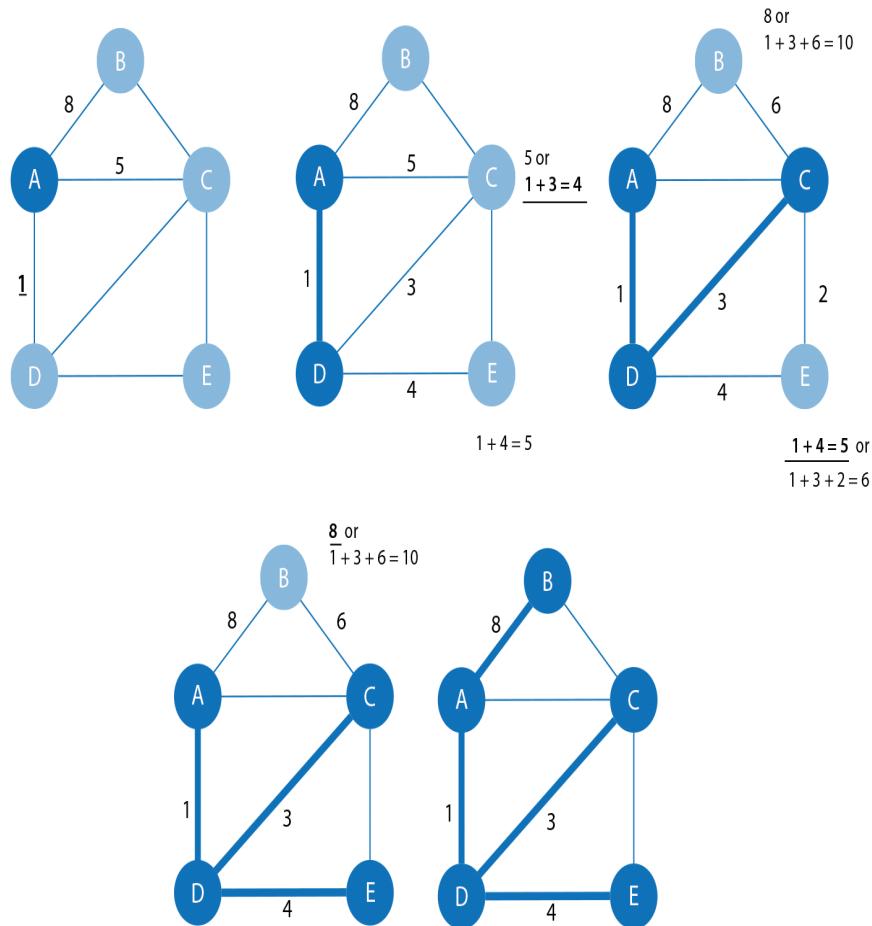


Figura 4-9. Los pasos del algoritmo de ruta más corta de una sola fuente.

Procede de la siguiente manera:

1. Comienza con un nodo raíz desde el cual se medirán todas las rutas. En la [Figura 4-9](#) hemos seleccionado el nodo A como la raíz.
2. La relación con el peso más pequeño proveniente de ese nodo raíz se selecciona y se agrega al árbol, junto con su nodo conectado. En este caso, eso es  $d(A, D) = 1$ .
3. La siguiente relación con el peso acumulativo más pequeño de nuestro nodo raíz a cualquier nodo no visitado se selecciona y se agrega al árbol de la misma manera. Nuestras opciones en la [Figura 4-9](#) son  $d(A, B) = 8$ ,  $d(A, C) = 5$  directamente o 4 a través de ADC, y  $d(A, E) = 5$ . Por lo tanto, se elige la ruta a través de ADC y se agrega C a nuestro árbol.

4. El proceso continúa hasta que no haya más nodos para agregar y tengamos nuestra ruta más corta de una sola fuente.

## ¿Cuándo debo usar la ruta más corta de una sola fuente?

Use la ruta más corta de una sola fuente cuando necesite evaluar la ruta óptima desde un punto de inicio fijo a todos los demás nodos individuales. Debido a que la ruta se elige en función del peso total de la ruta desde la raíz, es útil para encontrar la mejor ruta a cada nodo, pero no necesariamente cuando todos los nodos deben visitarse en un solo viaje.

Por ejemplo, el SSSP es útil para identificar las rutas principales que se deben utilizar para los servicios de emergencia en los que no visita cada ubicación en cada incidente, pero no para encontrar una ruta única para la recolección de basura en la que deba visitar cada casa en un solo viaje. (En este último caso, usaría el algoritmo del árbol de expansión mínimo, que se explicará más adelante).

Ejemplos de casos de uso incluyen:

- Detectar cambios en la topología, como fallas de enlaces, y [sugerir una nueva estructura de enrutamiento en segundos](#)
- Usar Dijkstra como un protocolo de enrutamiento IP para uso en sistemas autónomos como una [red de área local \(LAN\)](#).

## La ruta más corta de una sola fuente con Apache Spark

Podemos adaptar la función shortest\_path que escribimos para [calcular la ruta más corta entre dos ubicaciones](#) para que nos devuelva la ruta más corta de una ubicación a todas las demás. Tenga en cuenta que estamos utilizando de nuevo el marco de los mensajes agregados de Spark para personalizar nuestra función.

Primero importaremos las mismas bibliotecas que antes:

```
desde graphframes.lib importa AggregateMessages como AM desde pyspark.sql las funciones de importación como F
```

Y usaremos la misma función definida por el usuario para construir rutas:

```
add_path_udf = F.udf(ruta lambda , id : ruta + [ id ], ArrayType ( StringType () ))
```

Ahora para la función principal, que calcula la ruta más corta a partir de un origen:

```
def sssp ( g , origen , nombre_columna = "costo" ): vértices = g . vértices \. withColumn ( "visitada" , F . iluminado ( Falso )) \. withColumn ( "distancia" , F . , cuando ( g . vértices [ "id" ] == origen , 0 ) . otra manera ( float ( "inf" ))) \. withColumn ( "camino" , F . array ()) cached_vertices = AM . getCachedDataFrame ( vértices ) g2 = GraphFrame ( cached_vertices , g . bordes ) mientras g2 . vértices . filtro ( 'visitado == Falso' ) . primero (): current_node_id = g2 . vértices . filtrar ( 'visitó == Falso' ) . ordenar ( "distancia" ) . primero () . id msg_distance = AM . borde [ column_name ] + AM . src [ 'distancia' ] msg_path = add_path_udf ( AM . src [ "camino" ] , AM . src [ "id" ] ) msg_for_dst = F . cuando ( AM . src [ "id" ] == current_node_id , F . struct ( msg_distance , msg_path )) new_distances = g2 . aggregateMessages ( F . min ( AM . MSG ) . alias ( "aggMess" ), sendToDst = msg_for_dst ) new_visited_col = F . cuando ( g2 . vértices . visitó ! ( g2 . vértices . id == current_node_id ), True ) . de lo contrario ( Falso ) new_distance_col = F . cuando ( new_distances [ "aggMess" ] . isNotNull () & ( new_distances . aggMess [ "col1" ] < g2 . vértices . distance ), new_distances . aggMess [ "col1" ]) \. de lo contrario ( g2 . vértices . distancia ) new_path_col = F . cuando ( new_distances [ "aggMess" ] . isNotNull () & ( new_distances . aggMess [ "col1" ] < g2 . vértices . distance ), new_distances . aggMess [ "col2" ] . cast ( "array <string>" )) \. de lo contrario ( g2 . vértices . ruta ) new_vertices = g2 . vértices . join ( new_distances , on = "id" , how = "left_outer" ) \. drop ( new_distances [ "id" ]) \. withColumn ( "visited" , new_visited_col ) \. withColumn ( "newDistance" , new_distance_col ) \. withColumn ( "newPath" , new_path_col ) \. drop ( "aggMess" , "distance" , "path" ) \. withColumnRenamed ( 'newDistance' , 'distance' ) \. withColumnRenamed ( 'newPath' , 'path' ) cached_new_vertices = AM . getCachedDataFrame ( new_vertices ) g2 = GraphFrame ( cached_new_vertices , g2 . vértices \. withColumn ( "newPath" , add_path_udf ( "path" , "id" )) \. drop ( "visitado" , "ruta" ) \. withColumnRenamed ( "newPath" , "path" )
```

Si queremos encontrar el camino más corto desde Ámsterdam a todas las demás ubicaciones, podemos llamar a la función así:

```
via_udf = F.udf(ruta lambda : ruta [ 1 :- 1 ], ArrayType ( StringType () ))
```

```
result = sssp( g , "Amsterdam" , "cost" ) ( resultado . withColumn( "via" , via_udf( "path" )) . select( "id" , "distance" , "via" ) . sort( "distance" ) . show( truncate = False ) )
```

Definimos otra función definida por el usuario para filtrar los nodos de inicio y fin de la ruta resultante. Si ejecutamos ese código veremos el siguiente resultado:

#### carné de identidad distancia vía

Amsterdam	0.0	[]
Utrecht	46.0	[]
Den Haag	59.0	[]
Gouda	81.0	[Utrecht]
Rotterdam	85.0	[Den Haag]
Hoek van Holland	86.0	[Den Haag]
Felixstowe	293.0	[Den Haag, Hoek van Holland]
Ipswich	315.0	[Den Haag, Hoek van Holland, Felixstowe]
Colchester	347.0	[Den Haag, Hoek van Holland, Felixstowe, Ipswich]
Immingham	369.0	[]
Doncaster	443.0	[Immingham]
Londres	453.0	[Den Haag, Hoek van Holland, Felixstowe, Ipswich, Colchester]

En estos resultados, vemos las distancias físicas en kilómetros desde el nodo raíz, Ámsterdam, a todas las demás ciudades en el gráfico, ordenadas por la distancia más corta.

## La ruta más corta de una sola fuente con Neo4j

Neo4j implementa una variación de SSSP, llamada el [algoritmo Delta-Stepping](#) que divide el algoritmo de Dijkstra en una serie de fases que pueden ejecutarse en paralelo.

La siguiente consulta ejecuta el algoritmo Delta-Stepping:

```
MATCH (n: Place { id : "London" }) CALL algo.shortestPath.deltaStepping.stream( n , "distance" , 1.0 ) nodeId RENDIMIENTO,
distancia DONDE algo.isFinite( distance ) RETURN algo.getNodeById( nodeId ). id AS destino, distancia ORDEN POR distancia
```

La consulta devuelve el siguiente resultado:

destino	distancia
Londres	0.0
Colchester	106.0
Ipswich	138.0
Felixstowe	160.0
Doncaster	277.0
Immingham	351.0
Hoek van Holland	367.0
Den Haag	394.0
Rotterdam	400.0
Gouda	425.0
Ámsterdam	453.0
Utrecht	460.0

En estos resultados, vemos las distancias físicas en kilómetros desde el nodo raíz, Londres, a todas las demás ciudades en el gráfico, ordenadas por la distancia más corta.

## Árbol de expansión mínima

El algoritmo de árbol de expansión mínimo (peso) comienza desde un nodo determinado y encuentra todos los nodos accesibles y el conjunto de relaciones que conectan los nodos con el peso mínimo posible.

Atraviesa el siguiente nodo no visitado con el peso más bajo desde cualquier nodo visitado, evitando ciclos.

El primer algoritmo conocido de árbol de expansión de peso mínimo fue desarrollado por el científico checo Otakar Boruvka en 1926. El algoritmo de Prim, inventado en 1957, es el más simple y mejor conocido.

El algoritmo de Prim es similar al algoritmo de ruta más corta de Dijkstra, pero en lugar de minimizar la longitud total de una ruta que termina en cada relación, minimiza la longitud de cada relación individualmente. A diferencia del algoritmo de Dijkstra, tolera las relaciones de peso negativo.

El algoritmo del árbol de expansión mínima funciona como se muestra en la [Figura 4-10](#).

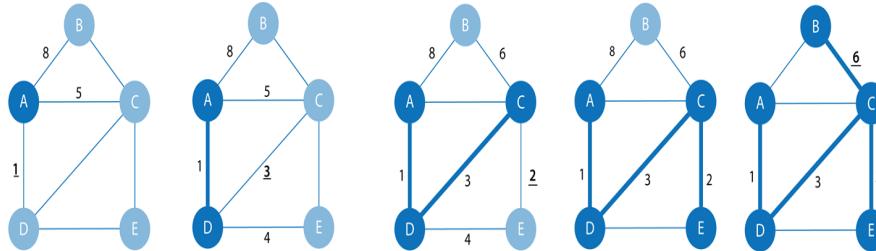


Figura 4-10. Los pasos del algoritmo de árbol de expansión mínima

Los pasos son los siguientes:

1. Comienza con un árbol que contiene un solo nodo. En la [Figura 4-10](#) comenzamos con el nodo A.
2. La relación con el peso más pequeño proveniente de ese nodo se selecciona y se agrega al árbol (junto con su nodo conectado). En este caso, AD.
3. Este proceso se repite, siempre eligiendo la relación de peso mínimo que se une a cualquier nodo que no esté en el árbol.
  - a. Si compara nuestro ejemplo aquí con el ejemplo de SSSP en la [Figura 4-9](#), notará que en la cuarta gráfica las rutas se vuelven diferentes. Esto se debe a que SSSP evalúa la ruta más corta basada en los totales acumulados desde la raíz, mientras que el Árbol de expansión mínima solo se ocupa del costo del siguiente paso.
4. Cuando no hay más nodos para agregar, el árbol es un árbol de expansión mínimo.

También hay variantes de este algoritmo que encuentran el árbol de expansión de peso máximo (árbol de costo más alto) y el árbol de distribución k (límite de tamaño de árbol).

## ¿Cuándo debo usar el árbol de expansión mínima?

Use el árbol de expansión mínima cuando necesite la mejor ruta para visitar todos los nodos. Debido a que la ruta se elige en función del costo de cada paso siguiente, es útil cuando debe visitar todos los nodos en un solo paseo. (Revise la sección anterior en ["Ruta más corta de una sola fuente"](#) si no necesita una ruta para un solo viaje).

Puede usar este algoritmo para optimizar rutas para sistemas conectados como tuberías de agua y diseño de circuitos. También se emplea para aproximar algunos problemas con tiempos de cálculo desconocidos, como el problema del vendedor ambulante y ciertos tipos de problemas de redondeo. Si bien puede que no siempre encuentre la solución óptima absoluta, este algoritmo hace que el análisis potencialmente complejo e intensivo de cómputo sea mucho más accesible.

Ejemplos de casos de uso incluyen:

- Minimizar el costo de viaje de explorar un país. [“Una aplicación de árboles de expansión mínimos para la planificación de viajes”](#) describe cómo el algoritmo analizó las conexiones aéreas y marítimas para hacer esto.
- Visualización de correlaciones entre rendimientos de moneda. Esto se describe en [“Aplicación del árbol de expansión mínima en el mercado de divisas”](#).
- [Rastrear el historial de transmisión de infecciones en un brote. Para obtener más información, consulte “Uso del modelo de árbol de expansión mínima para la investigación epidemiológica molecular de un brote nosocomial de infección por el virus de la hepatitis C”.](#)

### ADVERTENCIA

El algoritmo del árbol de expansión mínima solo proporciona resultados significativos cuando se ejecuta en un gráfico donde las relaciones tienen diferentes ponderaciones. Si la gráfica no tiene pesos, o todas las relaciones tienen el mismo peso, entonces cualquier árbol de expansión es un árbol de expansión mínima.

## Árbol de expansión mínimo con Neo4j

Veamos el algoritmo del árbol de expansión mínima en acción. La siguiente consulta encuentra un árbol de expansión a partir de Amsterdam:

```
MATCH (n: Place { id : "Amsterdam" }) CALL algo.spanningTree.minimum ( "Place" , "EROAD" , "distance" , id (n) , {write: true , writeProperty: "MINST" }) RENDIMIENTO loadMillis, computeMillis, writeMillis, effectNodeCount RETURN loadMillis, computeMillis, writeMillis, EffectNodeCount
```

Los parámetros pasados a este algoritmo son:

Lugar

Las etiquetas de nodo a considerar al calcular el árbol de expansión

EROAD

Los tipos de relación a considerar al calcular el árbol de expansión

distancia

El nombre de la propiedad de relación que indica el costo de atravesar entre un par de nodos

ID (n)

El ID de nodo interno del nodo desde el cual debe comenzar el árbol de expansión

Esta consulta almacena sus resultados en el gráfico. Si queremos devolver el árbol de expansión de peso mínimo, podemos ejecutar la siguiente consulta:

```
PARTIDO path = (n: Lugar { ID : "Amsterdam" }) - [: minst *] - (+) CON relaciones (path) AS REL Relájese REL como REL CON rel distintos como rel RETORNO startNode ( rel ). id fuente AS , endNode ( rel ). Identificación del AS destino, rel .distance AS coste
```

Y esta es la salida de la consulta:

fuente	destino	costo
Amsterdam	Utrecht	46.0
Utrecht	Gouda	35.0
Gouda	Rotterdam	25.0
Rotterdam	Den Haag	26.0
Den Haag	Hoek van Holland	27.0
Hoek van Holland	Felixstowe	207.0
Felixstowe	Ipswich	22.0
Ipswich	Colchester	32.0
Colchester	Londres	106.0

Londres	Doncaster	277.0
Doncaster	Immingham	74.0

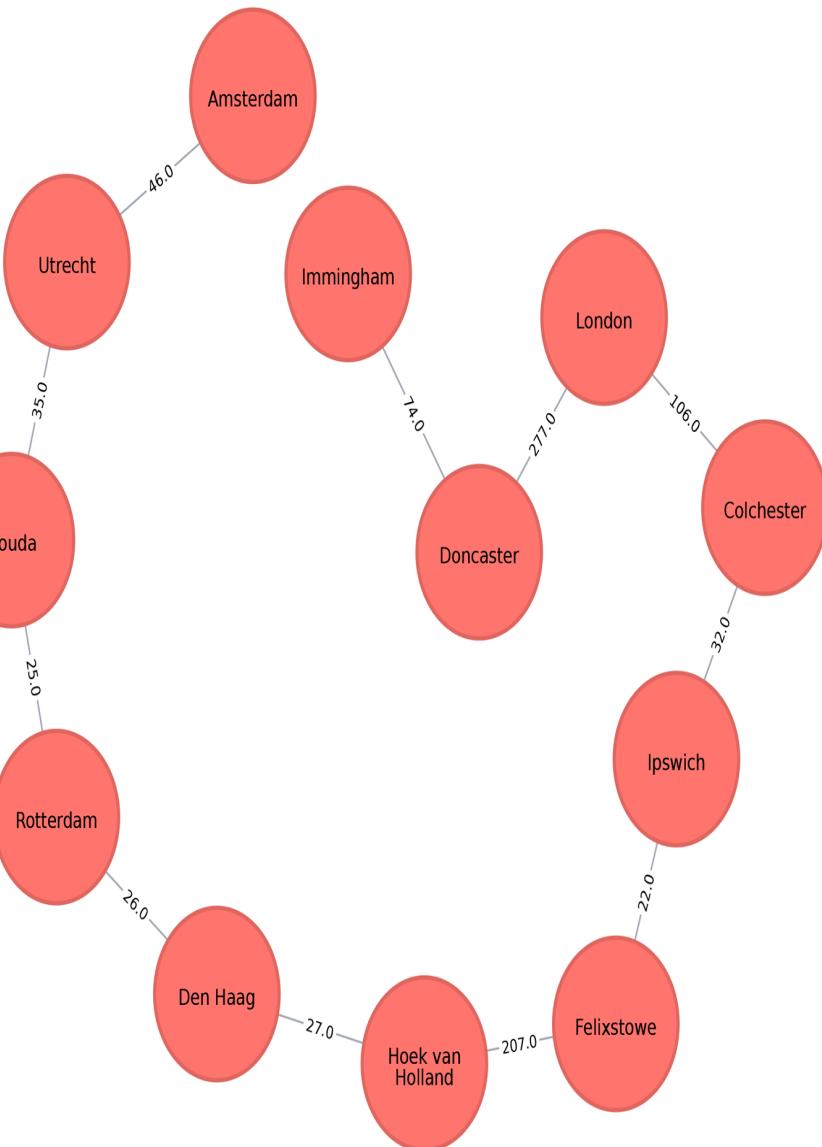


Figura 4-11. Un árbol de peso mínimo de Ámsterdam

Si estuviéramos en Ámsterdam y quisieramos visitar todos los demás lugares de nuestro conjunto de datos durante el mismo viaje, la [Figura 4-11](#) muestra la ruta continua más corta para hacerlo.

## Caminata aleatoria

El algoritmo de Paseo aleatorio proporciona un conjunto de nodos en una ruta aleatoria en un gráfico. El término fue mencionado por primera vez por Karl Pearson en 1905 en una carta a la revista Nature titulada "[El problema de la caminata aleatoria](#)". Aunque el concepto se remonta aún más, es solo más recientemente que se han aplicado caminatas aleatorias a la ciencia de redes.

Una caminata aleatoria, en general, a veces se describe como similar a cómo una persona ebria atraviesa una ciudad. Saben qué dirección o punto final desean alcanzar, pero pueden tomar una ruta muy tortuosa para llegar allí.

El algoritmo comienza en un nodo y de alguna manera sigue aleatoriamente una de las relaciones hacia adelante o hacia atrás a un nodo vecino. Luego hace lo mismo desde ese nodo y así sucesivamente, hasta que alcanza la longitud de ruta establecida. (Decimos algo al azar porque el número de relaciones que tiene un nodo y sus vecinos influyen en la probabilidad de que un nodo sea recorrido).

## ¿Cuándo debo usar la caminata aleatoria?

Utilice el algoritmo de paseo aleatorio como parte de otros algoritmos o líneas de datos cuando necesite generar un conjunto de nodos conectados en su mayoría al azar.

Ejemplos de casos de uso incluyen:

- Como parte de los algoritmos node2vec y graph2vec, que crean incrustaciones de nodo. Estas incrustaciones de nodo podrían usarse como entrada a una red neuronal.
- Como parte de la detección de la comunidad Walktrap e Infomap. Si una caminata aleatoria devuelve un pequeño conjunto de nodos repetidamente, entonces indica que el conjunto de nodos puede tener una estructura de comunidad.
- Como parte del proceso de formación de modelos de aprendizaje automático. Esto se describe más detalladamente en el artículo de David Mack ["Revisión de la predicción con Neo4j y TensorFlow"](#).

Puede leer sobre más casos de uso en un artículo de N. Masuda, MA Porter y R. Lambiotte, ["Random Walks and Diffusion on Networks"](#).

## Paseo aleatorio con Neo4j

Neo4j tiene una implementación del algoritmo Random Walk. Admite dos modos para elegir la siguiente relación a seguir en cada etapa del algoritmo:

aleatorio

Elige aleatoriamente una relación a seguir

node2vec

Elige la relación a seguir basándose en el cálculo de una distribución de probabilidad de los vecinos anteriores

La siguiente consulta hace esto:

```
MATCH (source: Place { id : "London" }) CALL algo.randomWalk.stream ( id (source), 5, 1) YIELD nodeId UNWIND
algo.getNodesById (nodeIds) AS place RETURN place. Identificación del AS lugar
```

Los parámetros pasados a este algoritmo son:

id (fuente)

El ID de nodo interno del punto de inicio para nuestra caminata aleatoria

5

El número de saltos que debe tomar nuestra caminata aleatoria.

1

El número de paseos aleatorios que queremos calcular.

Devuelve el siguiente resultado:

**lugar**

Londres

Doncaster

Immingham

Amsterdam

Utrecht

Amsterdam

En cada etapa de la caminata aleatoria, la siguiente relación se elige al azar. Esto significa que si ejecutamos el algoritmo, incluso con los mismos parámetros, es probable que no obtengamos el mismo resultado.

También es posible que un paseo regrese por sí mismo, como podemos ver en la [Figura 4-12](#), donde vamos de Amsterdam a Den Haag y regresamos.

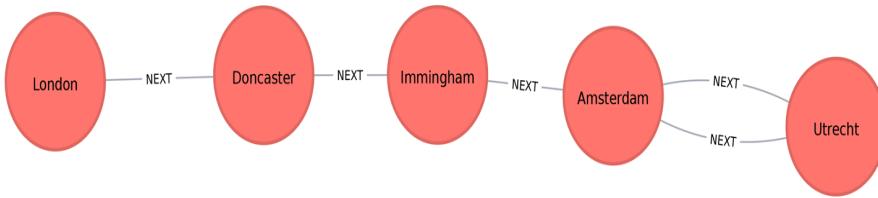


Figura 4-12. Un paseo aleatorio a partir de Londres

## Resumen

Los algoritmos de búsqueda de rutas son útiles para comprender la forma en que se conectan nuestros datos. En este capítulo, comenzamos con los algoritmos fundamentales de amplitud y profundidad primero, antes de pasar a Dijkstra y otros algoritmos de ruta más corta. También observamos las variantes de los algoritmos de ruta más corta optimizados para encontrar la ruta más corta de un nodo a todos los demás nodos o entre todos los pares de nodos en una gráfica. Terminamos con el algoritmo de paseo aleatorio, que se puede usar para encontrar conjuntos arbitrarios de rutas.

A continuación, aprenderemos sobre los algoritmos de centralidad que se pueden usar para encontrar nodos influyentes en un gráfico.

## RECURSO DE ALGORITMO

Hay muchos libros de algoritmos, pero uno se destaca por su cobertura de conceptos fundamentales y algoritmos de gráficos: *The Algorithm Design Manual*, de Steven S. Skiena (Springer). Recomendamos encarecidamente este libro de texto a aquellos que buscan un recurso completo sobre algoritmos clásicos y técnicas de diseño, o que simplemente deseen profundizar en cómo operan los distintos algoritmos.

# Capítulo 5. Algoritmos de centralidad.

Los algoritmos de centralidad se utilizan para entender los roles de los particulares. Los nodos en una gráfica y su impacto en esa red. Son útiles porque identifican los nodos más importantes y nos ayudan a comprender las dinámicas de los grupos, como la credibilidad, la accesibilidad, la velocidad a la que se propagan las cosas y los puentes entre los grupos. Aunque muchos de estos algoritmos fueron inventados para el análisis de redes sociales, desde entonces han encontrado usos en una variedad de industrias y campos.

Cubriremos los siguientes algoritmos:

- El grado de centralidad como una medida de línea de base de conectividad.
- Centralidad de proximidad para medir cuán central es un nodo para el grupo, incluidas dos variaciones para grupos desconectados
- Centralidad de intermediación para encontrar puntos de control, incluida una alternativa para la aproximación
- PageRank para comprender la influencia general, incluida una opción popular para la personalización

## PROPIA

Los diferentes algoritmos de centralidad pueden producir resultados significativamente diferentes según lo que se crearon para medir. Cuando vea respuestas subóptimas, es mejor verificar que el algoritmo que ha utilizado esté alineado con el propósito previsto.

Explicaremos cómo funcionan estos algoritmos y mostraremos ejemplos en Spark y Neo4j. Cuando un algoritmo no esté disponible en una plataforma o donde las diferencias no sean importantes, le brindaremos solo un ejemplo de plataforma.

[La Figura 5-1](#) muestra las diferencias entre los tipos de preguntas que los algoritmos de centralidad pueden responder, y la [Tabla 5-1](#) es una referencia rápida de lo que cada algoritmo calcula con un ejemplo de uso.

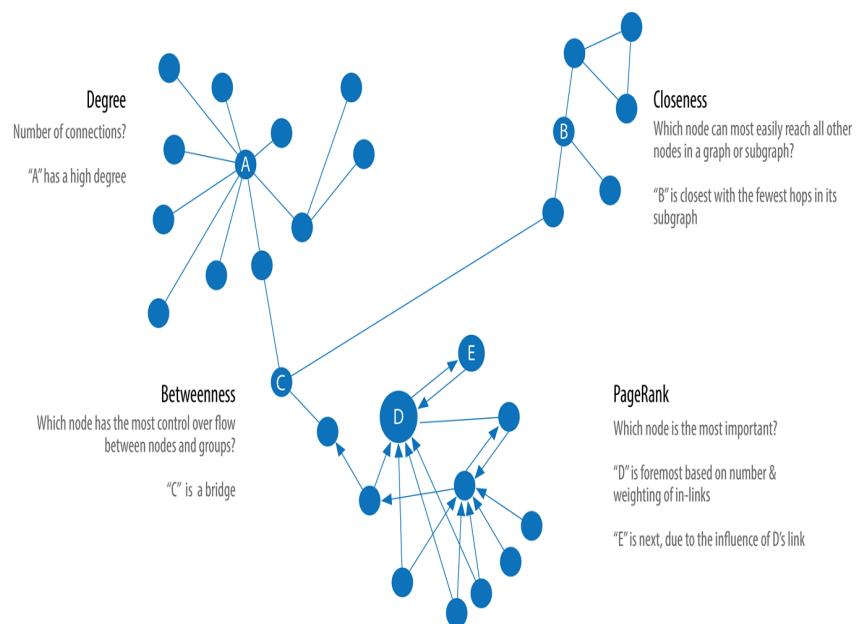


Figura 5-1. Los algoritmos de centralidad representativa y los tipos de preguntas que contestan.

Tabla 5-1. Visión general de los algoritmos de centralidad.

Tipo de algoritmo	Que hace	Ejemplo de uso	Ejemplo de chispa	Ejemplo neo4j
<a href="#">Grado de centralidad</a>	Mide el número de relaciones que tiene un nodo	Estimar la popularidad de una persona mirando su título y utilizando su grado para estimar la gregariedad	Sí	No
<a href="#">Proximidad</a>	Calcula qué nodos tienen las rutas	Encontrar la ubicación óptima de los nuevos servicios	Sí	Sí

[centralidad](#)

más cortas a todos los demás nodos públicos para la máxima accesibilidad.

[Variaciones:](#)[Wasserman y Fausto,](#)  
[Centralidad armónica.](#)[Centralidad de intermediaciación](#)[Variación:](#)[Brandes](#)[Aproximadas](#)[Aleatorizadas](#)

Mide el número de rutas más cortas que pasan a través de un nodo

Mejora de la selección de fármacos mediante la búsqueda de genes de control para enfermedades específicas

No

Sí

[Rango de página](#)[Variación:](#)  
[PageRank personalizado](#)

Estima la importancia de un nodo actual a partir de sus vecinos vinculados y sus vecinos (popularizado por Google)

Encontrar las características más influyentes para la extracción en el aprendizaje automático y clasificar el texto para la relevancia de la entidad en el procesamiento del lenguaje natural.

Sí

Sí

**NOTA**

Varios de los algoritmos de centralidad calculan las rutas más cortas entre cada par de nodos. Esto funciona bien para gráficos de tamaño pequeño a mediano, pero para gráficos grandes puede ser computacionalmente prohibitivo. Para evitar largos tiempos de ejecución en gráficos más grandes, algunos algoritmos (por ejemplo, la centralidad de la intermediación) tienen versiones aproximadas.

En primer lugar, describiremos el conjunto de datos de nuestros ejemplos y seguiremos importando los datos a Apache Spark y Neo4j. Cada algoritmo se trata en el orden listado en la [Tabla 5-1](#). Comenzaremos con una breve descripción del algoritmo y, cuando sea necesario, información sobre cómo funciona. Las variaciones de los algoritmos ya cubiertos incluirán menos detalles. La mayoría de las secciones también incluyen una guía sobre cuándo usar el algoritmo relacionado. Demostramos código de ejemplo usando un conjunto de datos de muestra al final de cada sección.

¡Empecemos!

## Ejemplo de datos de gráfico: El gráfico social

Los algoritmos de centralidad son relevantes para todos los gráficos, pero las redes sociales proporcionan una forma muy personal de pensar sobre la influencia dinámica y el flujo de información. Los ejemplos en este capítulo se ejecutan contra un pequeño gráfico similar a Twitter. Puede descargar los nodos y los archivos de relaciones que usaremos para crear nuestra gráfica desde el [repositorio GitHub del libro](#).

Tabla 5-2. social-nodes.csv

**carné de identidad**

Alicia

Bridget

Charles

Doug

marca

Miguel

David

Amy

James

Tabla 5-3. relaciones sociales.csv

**src dst relación**

Alicia	Bridget	Sigue
Alicia	Charles	Sigue
marca	Doug	Sigue
Bridget	Miguel	Sigue
Doug	marca	Sigue
Miguel	Alicia	Sigue
Alicia	Miguel	Sigue
Bridget	Alicia	Sigue
Miguel	Bridget	Sigue
Charles	Doug	Sigue
Bridget	Doug	Sigue
Miguel	Doug	Sigue
Alicia	Doug	Sigue
marca	Alicia	Sigue
David	Amy	Sigue
James	David	Sigue

[La figura 5-2](#) ilustra la gráfica que queremos construir.

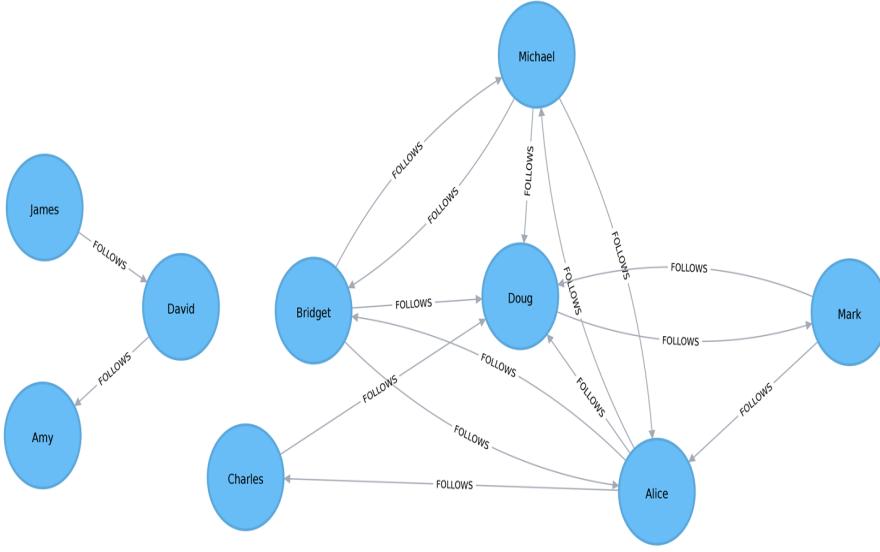


Figura 5-2. El modelo grafico

Tenemos un conjunto más grande de usuarios con conexiones entre ellos y un conjunto más pequeño sin conexiones a ese grupo más grande.

Vamos a crear gráficos en Spark y Neo4j basados en el contenido de esos archivos CSV.

## Importando los datos en Apache Spark

Primero, importaremos los paquetes requeridos de Spark y el paquete de GraphFrames:

```
desde graphframes import *
desde pyspark import SparkContext
```

Podemos escribir el siguiente código para crear un GraphFrame basado en el contenido de los archivos CSV:

```
v = chispa . lea . csv ( "data / social-nodes.csv" , header = True ) e = spark . lea . csv ( "data / social-relationships.csv" , header = True )
g = GraphFrame ( v , e )
```

## Importando los datos en Neo4j

A continuación, cargaremos los datos para Neo4j. La siguiente consulta importa nodos:

```
CON "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base WITH base + "social-nodes.csv" AS uri CSV DE
CARGA CON LOS JEFATORES DE uri AS fila MERGE (: Usuario { id : row.Id })
```

Y esta consulta importa relaciones:

```
CON "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base WITH base + "social-relationships.csv" AS uri CSV DE
CARGA CON LOS JEFATORES DE uri AS fila MATCH (fuente : Usuario { id : row.src }) MATCH (destino: Usuario { id : row.dst }) MERGE (fuente) - [: SIGUE] -> (destino)
```

Ahora que nuestros gráficos están cargados, ¡está en los algoritmos!

## Grado de centralidad

Degree Centrality es el más simple de los algoritmos que cubriremos en este libro. Cuenta el número de relaciones entrantes y salientes de un nodo, y se usa para encontrar nodos populares en un gráfico. El grado de centralidad fue propuesto por Linton C. Freeman en su artículo de 1979 "[Centralidad en las redes sociales: aclaración conceptual](#)" .

## Alcanzar

Comprender el alcance de un nodo es una medida justa de importancia. ¿Cuántos otros nodos puede tocar ahora? El grado de un nodo es el número de relaciones directas que tiene, calculadas para el grado y el grado. Puedes pensar en esto como el alcance inmediato del nodo. Por ejemplo, una persona con un alto grado en una red social activa tendría muchos contactos inmediatos y sería más probable que detectara un frío que circula en su red.

El grado promedio de una red es simplemente el número total de relaciones dividido por el número total de nodos; Puede estar muy sesgado por nodos de alto grado. La distribución de grados es la probabilidad de que un nodo seleccionado al azar tenga un cierto número de relaciones.

[La Figura 5-3](#) ilustra la diferencia al observar la distribución real de las conexiones entre los temas de subreddit. Si simplemente tomara el promedio, asumiría que la mayoría de los temas tienen 10 conexiones, mientras que, de hecho, la mayoría de los temas solo tienen 2 conexiones.

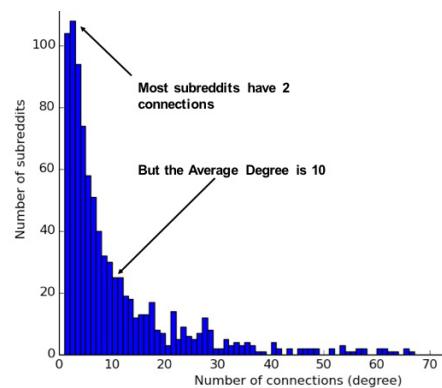


Figura 5-3. Este mapeo de la distribución del grado subreddit por [Jacob Silterra](#) proporciona un ejemplo de cómo el promedio no suele reflejar la distribución real en las redes. CC BY-SA 3.0.

Estas medidas se utilizan para categorizar los tipos de red, como las redes sin escala o de mundo pequeño que se analizaron en el [Capítulo 2](#). También proporcionan una medida rápida para ayudar a estimar el potencial de propagación o ondulación en una red.

## ¿Cuándo debo usar la centralidad del grado?

Utilice Grado Centralidad si está intentando analizar la influencia observando el número de relaciones entrantes y salientes, o encuentre la "popularidad" de los nodos individuales. Funciona bien cuando le

preocupa la conexión inmediata o las probabilidades a corto plazo. Sin embargo, el grado de centralidad también se aplica al análisis global cuando se desea evaluar el grado mínimo, el grado máximo, el grado medio y la desviación estándar en todo el gráfico.

Ejemplos de casos de uso incluyen:

- Identificar individuos poderosos a través de sus relaciones, como las conexiones de personas en una red social. Por ejemplo, en BrandWatch "[Los hombres y mujeres más influyentes en Twitter 2017](#)" , las 5 mejores personas de cada categoría tienen más de 40 millones de seguidores cada una.
- Separar a los estafadores de usuarios legítimos de un sitio de subastas en línea. La centralidad ponderada de los estafadores tiende a ser significativamente mayor debido a la colusión dirigida a aumentar artificialmente los precios. Lea más en el documento de P. Bangcharoensap et al., "[Aprendizaje semi supervisado basado en gráficos de dos pasos para la detección de fraudes en subastas en línea](#)" .

## Grado de Centralidad con Apache Spark

Ahora ejecutaremos el algoritmo Degree Centrality con el siguiente código:

```
total_degree = g . grados en grado = g . inDegrees out_degree = g . outDegrees ( total_degree . join ( in_degree , "id" , how = "left" ) . join ( out_degree , "id" , how = "left" ) . fillna ( 0 ) . sort ( "inDegree" , ascendente = Falso ) . mostrar () )
```

Primero calculamos los grados totales, de entrada y salida. Luego, unimos esos DataFrames juntos, usando una unión izquierda para retener cualquier nodo que no tenga relaciones entrantes o salientes. Si los nodos no tienen relaciones, establecemos ese valor en 0 utilizando la función fillna .

Aquí está el resultado de ejecutar el código en pyspark:

**carné de identidad la licenciatura en Grado fuera de grado**

Doug	6	5	1
Alicia	7	3	4
Miguel	5	2	3
Bridget	5	2	3
Charles	2	1	1
marca	3	1	2
David	2	1	1
Amy	1	1	0
James	1	0	1

Podemos ver en la [Figura 5-4](#) que Doug es el usuario más popular en nuestro gráfico de Twitter, con cinco seguidores (in-links). Todos los demás usuarios en esa parte del gráfico lo siguen y él solo sigue a una persona. En la red real de Twitter, las celebridades tienen un alto número de seguidores, pero tienden a seguir a pocas personas. ¡Por lo tanto podríamos considerar a Doug una celebridad!

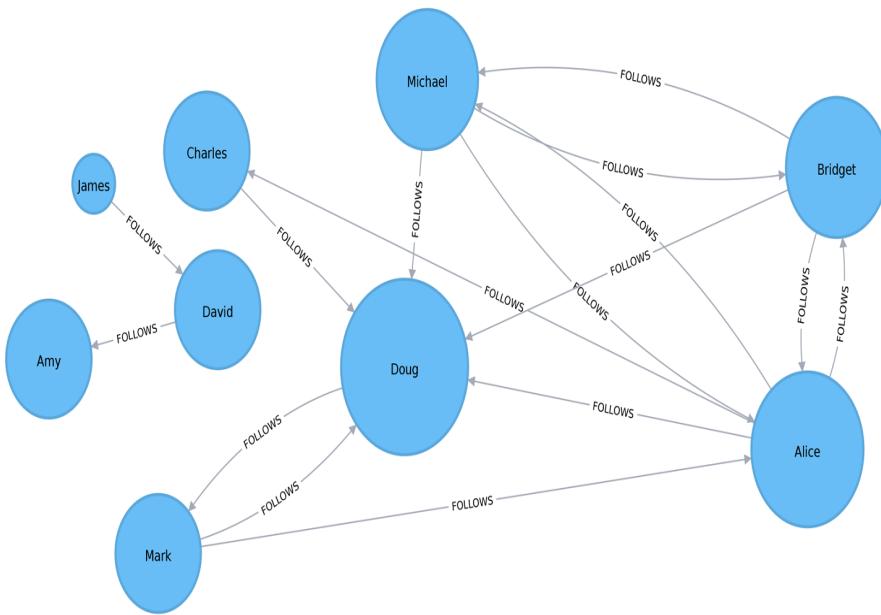


Figura 5-4. Visualización de la centralidad del grado.

Si estuviéramos creando una página que muestre a los usuarios más seguidos o quisiéramos sugerirle a otras personas, podríamos usar este algoritmo para identificar a esas personas.

### PROPIA

Algunos datos pueden contener nodos muy densos con muchas relaciones. Estos no agregan mucha información adicional y pueden sesgar algunos resultados o agregar complejidad computacional. Es posible que desee filtrar estas densas notas utilizando un subgrafo, o usar una proyección para resumir las relaciones como pesos.

## Proximidad centralidad

La proximidad centralidad es una forma de detectar nodos que pueden difundir información de manera eficiente a través de un subgrafo.

La medida de la centralidad de un nodo es su distancia promedio (distancia inversa) a todos los demás nodos. Los nodos con una puntuación de proximidad alta tienen las distancias más cortas de todos los demás nodos.

Para cada nodo, el algoritmo de centralidad de proximidad calcula la suma de sus distancias a todos los demás nodos, basándose en el cálculo de las rutas más cortas entre todos los pares de nodos. La suma resultante se invierte para determinar la puntuación de centralidad de proximidad para ese nodo.

La centralidad de proximidad de un nodo se calcula utilizando la fórmula:

$$C(u) = \frac{1}{N} \sum_{v=1}^{N-1} d(u, v)$$

dónde:

- U es un nodo.
- N es el número de nodos en el gráfico.
- D (u, v) es la distancia de la ruta más corta entre otro nodo v y u .

Es más común normalizar esta puntuación para que represente la longitud promedio de las rutas más cortas en lugar de su suma. Este ajuste permite comparaciones de la centralidad de proximidad de nodos de gráficos de diferentes tamaños.

La fórmula para la centralidad de la proximidad normalizada es la siguiente:

$$\text{Norma } C(u) = \frac{1}{N-1} \sum_{v=1}^{N-1} d(u, v)$$

## ¿Cuándo debo usar la centralidad de proximidad?

Aplique la centralidad de proximidad cuando necesite saber qué nodos diseminan las cosas más rápido. El uso de relaciones ponderadas puede ser especialmente útil para evaluar las velocidades de interacción en la comunicación y los análisis de comportamiento.

## Ejemplos de casos de uso incluyen:

- Descubrir individuos en posiciones muy favorables para controlar y adquirir información y recursos vitales dentro de una organización. Uno de esos estudios es "[Mapeo de redes de células terroristas](#)" , por VE Krebs.
- Como heurístico para estimar el tiempo de llegada en las telecomunicaciones y la entrega de paquetes, donde el contenido fluye a través de las rutas más cortas hacia un objetivo predefinido. También se utiliza para arrojar luz sobre la propagación a través de todos los caminos más cortos simultáneamente, como infecciones que se propagan a través de una comunidad local. Encuentre más detalles en "[Centralidad y flujo de red](#)" , por SP Borgatti.
- Evaluar la importancia de las palabras en un documento, basado en un proceso de extracción de frase clave basada en el gráfico. F. Boudin describe este proceso en "[Una comparación de medidas de centralidad para la extracción de frases clave basadas en gráficos](#)" .

## ADVERTENCIA

La proximidad centralidad funciona mejor en gráficos conectados. Cuando la fórmula original se aplica a un gráfico no conectado, terminamos con una distancia infinita entre dos nodos donde no hay una ruta entre ellos. Esto significa que terminaremos con una puntuación de centralidad de cercanía infinita cuando resumamos todas las distancias desde ese nodo. Para evitar este problema, una variación en la fórmula original se mostrará después del siguiente ejemplo.

## Centralidad de proximidad con chispa de apache

Apache Spark no tiene un algoritmo incorporado para la proximidad central, pero podemos escribir el nuestro usando el marco de agregación de mensajes que introdujimos en la "[Ruta más corta \(ponderada\) con Apache Spark](#)" en el capítulo anterior.

Antes de crear nuestra función, importaremos algunas bibliotecas que usaremos:

```
desde graphframes.lib importar AggregateMessages como AM desde pyspark.sql importar funciones como F desde pyspark.sql.types
importar * desde operador import itemgetter
```

También crearemos algunas funciones definidas por el usuario que necesitaremos más adelante:

```
def collect_paths ( caminos ): volver F . collect_set ( caminos ) collect_paths_udf = F . udf ( collect_paths , ArrayType ( StringType () ) )
paths_type = ArrayType ( StructType ([ StructField ( "id" , StringType () ), StructField ( "distance" , IntegerType () ) ]) )
def flatten ( ids ):
    flat_list = [ item de lista secundaria en los identificadores de elemento en la lista
secundaria ] retorno de la lista ( dict ( ordenados ( flat_list , clave = itemgetter ( 0 ) ) ) . artículos () ) flatten_udf = F . udf ( flatten ,
paths_type ) def new_paths ( rutas , id ): rutas = [{ "id" : col1 , "distance" : col2 + 1 } para col1 , col2 en rutas si col1 != id ]
rutas . append ({ "id" : Identificación , "distancia" : 1 }) de regreso caminos new_paths_udf = F . udf ( new_paths , paths_type ) def
merge_paths ( ids , new_ids , id ): joined_ids = ids + ( new_ids if new_ids else [] ) merged_ids = [( col1 ,
col2 ) para col1 , col2 en joined_ids si col1 != id ] best_ids = dict ( ordenados ( merged_ids , key = itemgetter ( 1 ), reversa = Verdadero
)) devuelve [{ "id" : col1 , "distance" : col2 } para col1 , col2 en best_ids . artículos ()] merge_paths_udf = F . udf (
merge_paths , paths_type ) def calcular_closeness ( ids ): nodos = len ( ids ) total_distance = sum ( [ col2 para col1 , col2 en ids ] )
devolver 0 si total_distance == 0 demás nodos * 1,0 / total_distance closeness_udf = F . UDF ( calculate_closeness ,
DoubleType () )
```

Y ahora, para el cuerpo principal que calcula la centralidad de proximidad para cada nodo:

```
vértices = g . vértices . withColumn ( "ID" , F . array () ) cached_vertices = AM . getCachedDataFrame ( vértices ) g2 = GraphFrame (
cached_vertices , g . bordes ) para i en rango ( 0 , g2 . vértices . contar () ): msg_dst = new_paths_udf ( AM . src [ "ids" ], AM . src [
"id" ] ) msg_src = new_paths_udf ( AM . dst [ "ids" ], AM . dst [ "id" ] ) agg = g2 . aggregateMessages ( F . collect_set ( AM . MSG ) .
alias ( "agg" ), sendToSrc = msg_src , sendToDst = msg_dst ) res = agg . withColumn ( "newIds" , flatten_udf ( "agg" ) ) . drop ( "agg" ) new_vertices = ( g2 . vertices . join ( res , on = "id" , how = "left_outer" ) . withColumn ( "mergedIds" ,
merge_paths_udf ( "ids" , "newIds" , "id" ) drop ( "ids" , "newIds" ) ) . withColumnRenamed ( "mergedIds" , "ids" ) )
cached_new_vertices = AM . getCachedDataFrame ( new_vertices ) g2 = GraphFrame ( cached_new_vertices ,
g2 . bordes ) ( g2 . vértices . withColumn ( "proximidad" , closeness_udf ( "ID" ) ) . especie ( " " , ascendiendo = falso ) ) . mostrar (
truncar = falso )
```

Si lo ejecutamos veremos el siguiente resultado:

carné de identidad ids	cercanía
Doug	[[Charles, 1], [Mark, 1], [Alice, 1], [Bridget, 1], [Michael, 1]] 1.0

Alicia	[[Charles, 1], [Mark, 1], [Bridget, 1], [Doug, 1], [Michael, 1]]	1.0
David	[[James, 1], [Amy, 1]]	1.0
Bridget	[[[Charles, 2], [Mark, 2], [Alice, 1], [Doug, 1], [Michael, 1]]]	0.7142857142857143
Miguel	[[[Charles, 2], [Mark, 2], [Alice, 1], [Doug, 1], [Bridget, 1]]]	0.7142857142857143
James	[[[Amy, 2], [David, 1]]]	0.6666666666666666
Amy	[[[Santiago, 2], [David, 1]]]	0.6666666666666666
marca	[[[Bridget, 2], [Charles, 2], [Michael, 2], [Doug, 1], [Alice, 1]]]	0.625
Charles	[[[Bridget, 2], [Mark, 2], [Michael, 2], [Doug, 1], [Alice, 1]]]	0.625

Alice, Doug y David son los nodos más estrechamente conectados en el gráfico con una puntuación de 1.0, lo que significa que cada uno se conecta directamente a todos los nodos en su parte del gráfico. [La Figura 5-5](#) ilustra que aunque David solo tiene algunas conexiones, dentro de su grupo de amigos, eso es significativo. En otras palabras, esta puntuación representa la cercanía de cada usuario a otros dentro de su subgrafo pero no la gráfica completa.

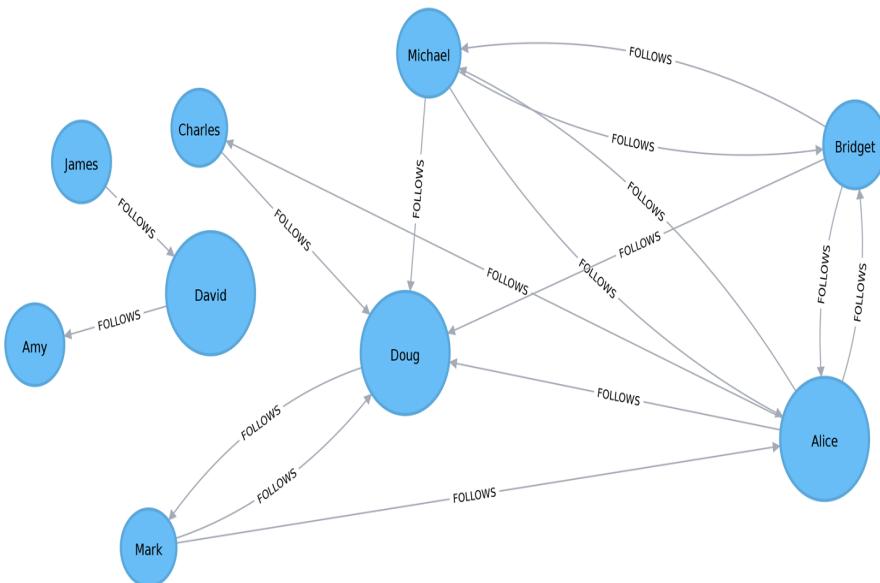


Figura 5-5. Visualización de la centralidad de la cercanía.

## Centralidad de proximidad con Neo4j

La implementación de Neo4j de Closeness Centrality usa la siguiente fórmula:

$$C(u) = \frac{n-1}{n-1} \sum_{v=1}^{n-1} \frac{1}{d(u,v)}$$

dónde:

- U es un nodo.
- N es el número de nodos en el mismo componente (subgrafo o grupo) que u .
- D (u, v) es la distancia de la ruta más corta entre otro nodo v y u .

Una llamada al siguiente procedimiento calculará la centralidad de proximidad para cada uno de los nodos de nuestro gráfico:

```
LLAME algo.closeness.stream ( "Usuario" , " SIGUE " ) IDENTIFICACIÓN DE NIVEL DE IDENTIFICACIÓN , centralidad
DEVOLUCIÓN algo.getNodeById (nodeId).id , centralidad ORDEN POR centralidad DESC
```

Ejecutando este procedimiento da el siguiente resultado:

**usuario centralidad**

Alicia	1.0
--------	-----

Doug	1.0
David	1.0
Bridget	0.7142857142857143
Miguel	0.7142857142857143
Amy	0.6666666666666666
James	0.6666666666666666
Charles	0.625
marca	0.625

Obtenemos los mismos resultados que con el algoritmo Spark, pero, como antes, la puntuación representa su cercanía con los demás dentro de su subgrafo pero no la gráfica completa.

### NOTA

En la interpretación estricta del algoritmo de centralidad de proximidad, todos los nodos de nuestra gráfica tendrían una puntuación de  $\infty$  porque cada nodo tiene al menos otro nodo al que no se puede alcanzar. Sin embargo, suele ser más útil implementar la puntuación por componente.

Idealmente, nos gustaría obtener una indicación de proximidad en todo el gráfico, y en las siguientes dos secciones aprenderemos sobre algunas variaciones del algoritmo de Centralidad de Cercanía que hacen esto.

## Variación de la proximidad de la centralidad: Wasserman y Fausto

Stanley Wasserman y Katherine Faust idearon una fórmula mejorada para calcular la cercanía de los gráficos con múltiples subgrafos sin conexiones entre esos grupos. Los detalles sobre su fórmula se encuentran en su libro, Análisis de redes sociales: Métodos y aplicaciones . El resultado de esta fórmula es una proporción de la fracción de nodos en el grupo que se puede alcanzar a la distancia promedio de los nodos alcanzables.

La fórmula es la siguiente:

$$C_{WF}(u) = \frac{n-1}{N-1} \sum_{v=1}^{n-1} \frac{1}{n-1} d(u, v)$$

dónde:

- U es un nodo.
- N es el recuento total de nodos.
- n es el número de nodos en el mismo componente que u .
- D (u, v) es la distancia de la ruta más corta entre otro nodo v y u .

Podemos decirle al procedimiento de Centralidad de proximidad que use esta fórmula al pasar el parámetro mejorado: verdadero .

La siguiente consulta ejecuta la Centralidad de Cercanía utilizando la fórmula de Wasserman y Faust:

```
CALL algo.closeness.stream ( "Usuario" , "SIGUE" , {mejorado: verdadero } ) ID de nodo RENDIMIENTO, centralidad RETORNO
algo.getNodeById (nodeId).id AS usuario, centralidad ORDEN POR centralidad DESC
```

El procedimiento da el siguiente resultado:

<b>usuario centralidad</b>	
Alicia	0.5
Doug	0.5
Bridget	0.35714285714285715
Miguel	0.35714285714285715
Charles	0.3125
marca	0.3125
David	0.125

Amy	0.0833333333333333
-----	--------------------

James	0.0833333333333333
-------	--------------------

Como muestra la [Figura 5-6](#), los resultados ahora son más representativos de la proximidad de los nodos a todo el gráfico. Las puntuaciones de los miembros del subgrafo más pequeño (David, Amy y James) se han reducido y ahora tienen las puntuaciones más bajas de todos los usuarios. Esto tiene sentido ya que son los nodos más aislados. Esta fórmula es más útil para detectar la importancia de un nodo en todo el gráfico que en su propio subgrafo.

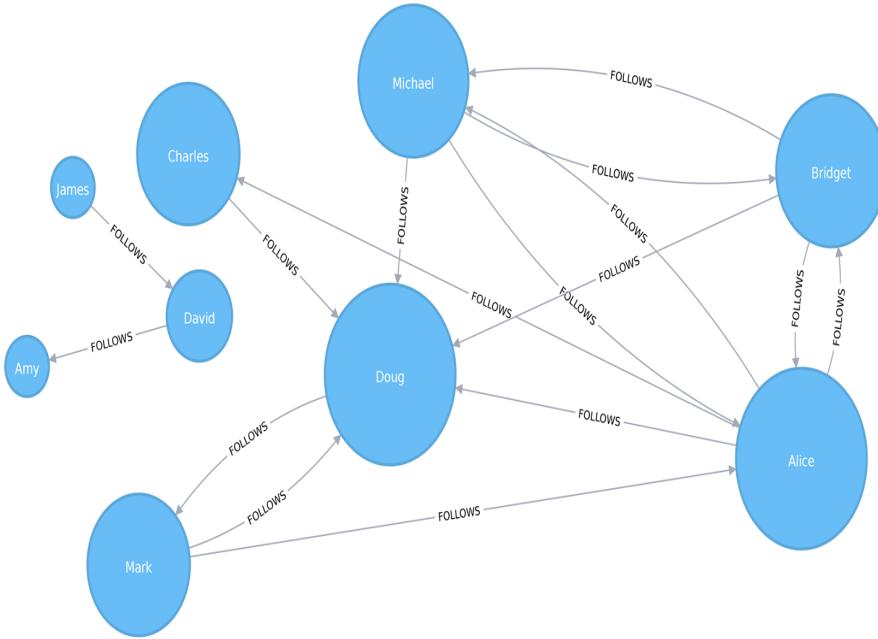


Figura 5-6. Visualización de la centralidad de la cercanía.

En la siguiente sección aprenderemos sobre el algoritmo de Centralidad Armónica, que logra resultados similares utilizando otra fórmula para calcular la cercanía.

## Variación de la proximidad de la centralidad: Centralidad armónica

La centralidad armónica (también conocida como centralidad valorada) es una variante de la proximidad central, inventada para resolver el problema original con gráficos no conectados. En [“Armonía en un mundo pequeño”](#), M. Marchiori y V. Latora propusieron este concepto como una representación práctica de un camino promedio más corto.

Cuando se calcula la puntuación de proximidad para cada nodo, en lugar de sumar las distancias de un nodo a todos los demás nodos, se suma la inversa de esas distancias. Esto significa que los valores infinitos se vuelven irrelevantes.

La centralidad armónica en bruto para un nodo se calcula utilizando la siguiente fórmula:

$$H(u) = \sum_{v=1}^{n-1} \frac{1}{d(u,v)}$$

dónde:

- U es un nodo.
- N es el número de nodos en el gráfico.
- D (u, v) es la distancia de la ruta más corta entre otro nodo v y u .

Al igual que con la centralidad de proximidad, también podemos calcular una centralidad armónica normalizada con la siguiente fórmula:

$$\text{Norma } H(u) = \sum_{v=1}^{n-1} \frac{1}{d(u,v)} \cdot \frac{n-1}{n}$$

En esta fórmula, los valores de  $\infty$  se manejan limpiamente.

## Centralidad armónica con Neo4j

La siguiente consulta ejecuta el algoritmo de Centralidad Armónica:

```
LLAME algo.closeness.harmonic.stream ( "Usuario" , " SIGUE " ) IDENTIFICACIÓN DE NIVEL DE IDENTIFICACIÓN ,
centralidad DEVOLUCIÓN algo.getNodeById (nodeId). id AS usuario, centralidad ORDEN POR centralidad DESC
```

Ejecutar este procedimiento da el siguiente resultado:

#### **usuario centralidad**

Alicia	0.625
Doug	0.625
Bridget	0.5
Miguel	0.5
Charles	0.4375
marca	0.4375
David	0.25
Amy	0.1875
James	0.1875

Los resultados de este algoritmo difieren de los del algoritmo original de Centralidad de Cercanía, pero son similares a los de la mejora de Wasserman y Fausto. Se puede usar cualquiera de los dos algoritmos cuando se trabaja con gráficos con más de un componente conectado.

## **Centralidad de intermediación**

A veces, el engranaje más importante del sistema no es el que tiene la mayor potencia manifiesta o el estado más alto. A veces, los intermediarios que conectan a los grupos o los intermediarios son quienes más controlan los recursos o el flujo de información. Intermediación La centralidad es una forma de detectar la cantidad de influencia que un nodo tiene sobre el flujo de información o recursos en una gráfica. Normalmente se usa para encontrar nodos que sirven de puente de una parte de una gráfica a otra.

El algoritmo de centralidad de intermediación primero calcula la ruta más corta (ponderada) entre cada par de nodos en un gráfico conectado. Cada nodo recibe una puntuación, según el número de estas rutas más cortas que pasan a través del nodo. Cuantas más rutas más cortas tenga un nodo, mayor será su puntuación.

La centralidad de la intermediación fue considerada una de las "tres concepciones intuitivas distintas de centralidad" cuando fue presentada por Linton C. Freeman en su artículo de 1971, "[Un conjunto de medidas de centralidad basadas en la dualidad](#)" .

### **Puentes y puntos de control.**

Un puente en una red puede ser un nodo o una relación. En un gráfico muy simple, puede encontrarlos buscando el nodo o la relación que, de eliminarse, provocaría que una sección del gráfico se desconecte. Sin embargo, como no es práctico en un gráfico típico, usamos un algoritmo de centralidad de intermediación. También podemos medir la intermediación de un clúster al tratar al grupo como un nodo.

Un nodo se considera fundamental para otros dos nodos si se encuentra en cada ruta más corta entre esos nodos, como se muestra en la [Figura 5-7](#) .

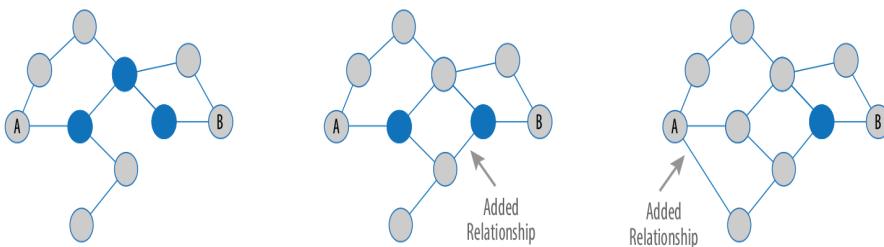


Figura 5-7. Los nodos pivotales se encuentran en cada ruta más corta entre dos nodos. La creación de rutas más cortas puede reducir la cantidad de nodos centrales para usos como la mitigación de riesgos.

Los nodos pivotes desempeñan un papel importante en la conexión de otros nodos: si elimina un nodo pivotal, la nueva ruta más corta para los pares de nodos originales será más larga o más costosa. Esto puede ser una consideración para evaluar puntos únicos de vulnerabilidad.

### Cálculo de la centralidad de intermediación

La centralidad de intermediación de un nodo se calcula agregando los resultados de la siguiente fórmula para todas las rutas más cortas:

$$B(u) = \sum_{s \neq u \neq t} p(u)p$$

dónde:

- U es un nodo.
- P es el número total de rutas más cortas entre los nodos s y t .
- P (u) es el número de rutas más cortas entre los nodos s y t que pasan a través del nodo u .

[La Figura 5-8](#) ilustra los pasos para trabajar con la centralidad de la intermediación.

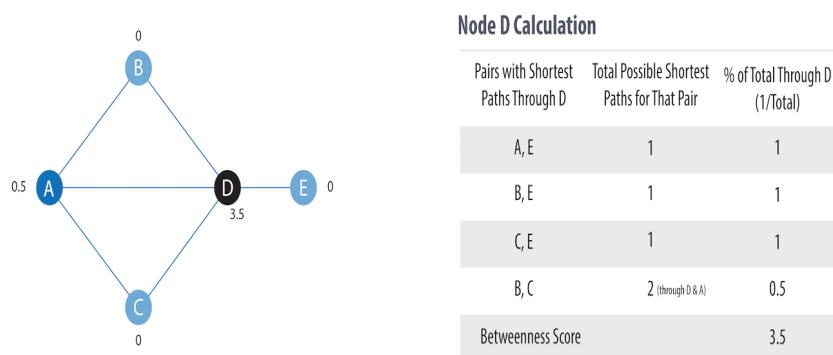


Figura 5-8. Conceptos básicos para calcular la centralidad de la intermediación.

Aquí está el procedimiento:

1. Para cada nodo, encuentre las rutas más cortas que lo atraviesan.
  - a. B, C, E no tienen rutas más cortas y se les asigna un valor de 0.
2. Para cada ruta más corta en el paso 1, calcule su porcentaje del total de las rutas más cortas posibles para ese par.
3. Sume todos los valores en el paso 2 para encontrar la puntuación de centralidad de la intermediación de un nodo. La tabla en la [Figura 5-8](#) ilustra los pasos 2 y 3 para el nodo D.
4. Repita el proceso para cada nodo.

## ¿Cuándo debo usar la centralidad de intermediación?

La centralidad de intermediación se aplica a una amplia gama de problemas en las redes del mundo real. Lo usamos para encontrar cuellos de botella, puntos de control y vulnerabilidades.

Ejemplos de casos de uso incluyen:

- Identificación de personas influyentes en diversas organizaciones. Los individuos poderosos no están necesariamente en posiciones de administración, pero se pueden encontrar en "posiciones de intermediación" usando la centralidad de intermediación. La eliminación de tales personas influyentes puede desestabilizar seriamente la organización. Esto podría considerarse una interrupción bienvenida por parte de la policía si la organización es delictiva, o podría ser un desastre si una empresa pierde personal clave que subestimó. Más detalles se encuentran en "[Calificaciones de corretaje en operaciones de timbre](#)" , por C. Morselli y J. Roy.
- Descubrir puntos de transferencia clave en redes como las redes eléctricas. En contra de la intuición, la eliminación de puentes específicos en realidad puede mejorar la robustez general al "aislar" las perturbaciones. Los detalles de la investigación se incluyen en "[Robustez de las redes eléctricas europeas bajo ataque intencional](#)" , por R. Solé, et al.
- Ayudar a los microbloggers a difundir su alcance en Twitter, con un motor de recomendación para dirigirse a personas influyentes. Este enfoque se describe en un artículo de S. Wu et al., "[Realización de recomendaciones en un microblog para mejorar el impacto de un usuario focal](#)" .

## PROPINA

La centralidad de la interrelación supone que todas las comunicaciones entre nodos se producen en el camino más corto y con la misma frecuencia, lo que no siempre es así en la vida real. Por lo tanto, no nos da una visión perfecta de los nodos más influyentes en una gráfica, sino una buena representación. Mark Newman explica esto con más detalle en [Redes: una introducción](#) (Oxford University Press, p186).

## Centralidad de intermediación con Neo4j

Spark no tiene un algoritmo incorporado para la centralidad de intermediación, por lo que demostraremos este algoritmo utilizando Neo4j. Una llamada al siguiente procedimiento calculará la centralidad de intermediación para cada uno de los nodos de nuestro gráfico:

```
CALL algo.betweenness.stream ( "User" , "FOLLOWS" ) YIELD nodeId, centralidad RETURN algo.getNodeById (nodeId).id AS usuario, centralidad ORDER BY centralidad DESC
```

Ejecutar este procedimiento da el siguiente resultado:

usuario	centralidad
Alicia	10.0
Doug	7.0
marca	7.0
David	1.0
Bridget	0.0
Charles	0.0
Miguel	0.0
Amy	0.0
James	0.0

Como podemos ver en la [Figura 5-9](#) , Alice es el corredor principal en esta red, pero Mark y Doug no se quedan atrás. En el subgrafo más pequeño, todos los caminos más cortos pasan por David, por lo que es importante para el flujo de información entre esos nodos.

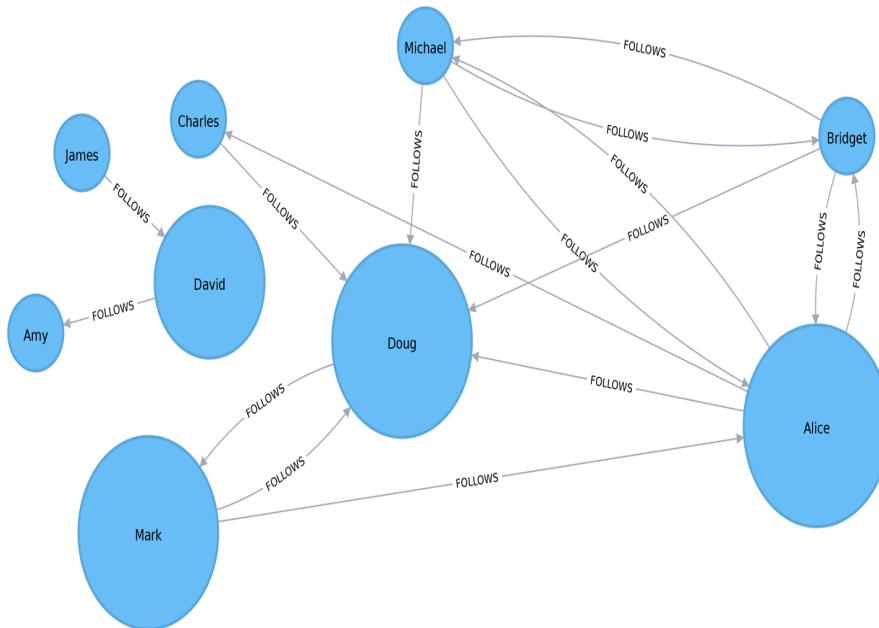


Figura 5-9. Visualización de la centralidad de intermediación.

### ADVERTENCIA

Para gráficos grandes, el cálculo de centralidad exacta no es práctico. El algoritmo más rápido conocido para calcular exactamente la interrelación de todos los nodos tiene un tiempo de ejecución proporcional al producto del número de nodos y el número de relaciones.

Es posible que queramos filtrar a un subgrafo primero o usar (que se describe en la siguiente sección) que funcione con un subconjunto de nodos.

Podemos unir nuestros dos componentes desconectados mediante la introducción de un nuevo usuario llamado Jason, que sigue y es seguido por personas de ambos grupos de usuarios:

```
CON [ "James" , "Michael" , "Alice" , "Doug" , "Amy" ] COMO SISTEMAS ACTUALES DE LOS USUARIOS (existentes: Usuario)
DONDE existe. id IN existingUsers MERGE (newUser: User { id : "Jason" }) MERGE (newUser) <- [: SIGUE] - (existente) MERGE
(newUser) - [: FOLLOWS] -> (existente)
```

Si volvemos a ejecutar el algoritmo veremos esta salida:

#### usuario centralidad

Jason	44.33333333333333
Doug	18.33333333333332
Alicia	16.666666666666664
Amy	8.0
James	8.0
Miguel	4.0
marca	2.166666666666665
David	0.5
Bridget	0.0
Charles	0.0

Jason tiene la puntuación más alta porque la comunicación entre los dos grupos de usuarios pasará a través de él. Se puede decir que Jason actúa como un puente local entre los dos grupos de usuarios, como se ilustra en la [Figura 5-10](#).

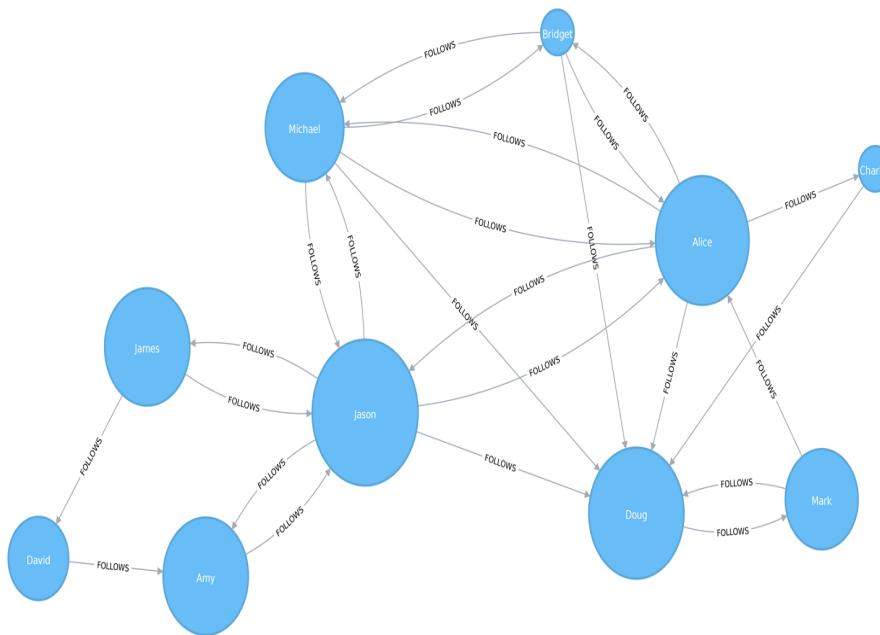


Figura 5-10. Visualización de la centralidad de intermediación con Jason.

Antes de pasar a la siguiente sección, restablecemos nuestra gráfica eliminando a Jason y sus relaciones:

PARTIDO (usuario: El usuario { ID : "Jason" }) SEPARA BORRAR usuario

## Variación de centralidad de la unidad: marcas aleatorias-aproximadas

Recuerde que el cálculo de la centralidad de intermediación exacta en gráficos grandes puede ser muy costoso. Por lo tanto, podríamos optar por utilizar un algoritmo de aproximación que se ejecute mucho más rápido pero que aún así proporcione información útil (aunque imprecisa).

El algoritmo de marcas aleatorias-aproximadas (RA-Brandes para abreviar) es el algoritmo más conocido para calcular una puntuación aproximada para la centralidad de intermediación. En lugar de calcular la ruta más corta entre cada par de nodos, el algoritmo RA-Brandes considera solo un subconjunto de nodos. Dos estrategias comunes para seleccionar el subconjunto de nodos son:

### Aleatorio

Los nodos se seleccionan de manera uniforme, al azar, con una probabilidad de selección definida. La probabilidad por defecto es:  $\log_{10}(N)/2$ . Si la probabilidad es 1, el algoritmo funciona de la misma manera que el algoritmo normal de centralidad de intermediación, donde se cargan todos los nodos.

### La licenciatura

Los nodos se seleccionan al azar, pero aquellos cuyo grado es menor que la media se excluyen automáticamente (es decir, solo los nodos con muchas relaciones tienen la posibilidad de ser visitados).

Como una optimización adicional, podría limitar la profundidad utilizada por el algoritmo de ruta más corta, que luego proporcionará un subconjunto de todas las rutas más cortas.

### Aproximación de la centralidad de intermediación con Neo4j

La siguiente consulta ejecuta el algoritmo RA-Brandes utilizando el método de selección aleatoria:

```
CALL algo.betweenness.sampled.stream( "User" , "FOLLOWS" , {strategy: "degree" } ) nodeId YIELD, centralidad RETURN algo.getNodeById( nodeId ).id AS usuario, centralidad ORDER BY centralidad DESC
```

Ejecutar este procedimiento da el siguiente resultado:

usuario centralidad

Alicia	9.0
marca	9.0
Doug	4.5
David	2.25
Bridget	0.0
Charles	0.0
Miguel	0.0
Amy	0.0
James	0.0

Nuestros mejores influencers son similares a los de antes, aunque Mark ahora tiene una clasificación más alta que Doug.

Debido a la naturaleza aleatoria de este algoritmo, podemos ver resultados diferentes cada vez que lo ejecutamos. En gráficos más grandes, esta aleatoriedad tendrá menos impacto que en nuestro pequeño gráfico de muestra.

## Rango de página

PageRank es el más conocido de los algoritmos de centralidad. Mide la influencia transitiva (o direccional) de los nodos. Todos los demás algoritmos de centralidad que analizamos miden la influencia directa de un nodo, mientras que PageRank considera la influencia de los vecinos de un nodo y sus vecinos. Por ejemplo, tener algunos amigos muy poderosos puede hacerte más influyente que tener muchos amigos menos poderosos. El PageRank se calcula distribuyendo de forma iterativa el rango de un nodo sobre sus vecinos o atravesando aleatoriamente el gráfico y contando la frecuencia con la que se golpea a cada nodo durante estas caminatas.

PageRank lleva el nombre del cofundador de Google, Larry Page, quien lo creó para clasificar los sitios web en los resultados de búsqueda de Google. El supuesto básico es que una página con más enlaces entrantes y más influyentes es más probable que sea una fuente creíble. PageRank mide el número y la calidad de las relaciones entrantes a un nodo para determinar una estimación de qué tan importante es ese nodo. Se supone que los nodos con más influencia sobre una red tienen más relaciones entrantes de otros nodos influyentes.

## Influencia

La intuición detrás de la influencia es que las relaciones con los nodos más importantes contribuyen más a la influencia del nodo en cuestión que las conexiones equivalentes a los nodos menos importantes. La medición de la influencia suele implicar nodos de puntuación, a menudo con relaciones ponderadas, y luego actualizar las puntuaciones en muchas iteraciones. A veces, todos los nodos se califican y, a veces, se utiliza una selección aleatoria como una distribución representativa.

### NOTA

Tenga en cuenta que las medidas de centralidad representan la importancia de un nodo en comparación con otros nodos. La centralidad es una clasificación del impacto potencial de los nodos, no una medida del impacto real. Por ejemplo, puede identificar a las dos personas con la centralidad más alta en una red, pero tal vez estén en juego políticas o normas culturales que en realidad cambien la influencia hacia otras. La cuantificación del impacto real es un área de investigación activa para desarrollar métricas de influencia adicionales.

## La fórmula de PageRank

PageRank se define en el documento original de Google de la siguiente manera:

$$\text{PR (u)} = (1 - d) + d (\text{PR (T1)} C (T1) + \dots + \text{PR (Tn)} C (Tn))$$

dónde:

- Suponemos que una página u tiene citas de las páginas T1 a Tn .

- D es un factor de amortiguamiento que se establece entre 0 y 1. Por lo general, se establece en 0,85. Puede pensar en esto como la probabilidad de que un usuario continúe haciendo clic. Esto ayuda a minimizar el hundimiento de rango, explicado en la siguiente sección.
- $1-d$  es la probabilidad de que un nodo se alcance directamente sin seguir ninguna relación.
- $C(T_n)$  se define como el grado de salida de un nodo T .

[La Figura 5-11](#) muestra un pequeño ejemplo de cómo PageRank continuará actualizando el rango de un nodo hasta que converja o cumpla con el número establecido de iteraciones.

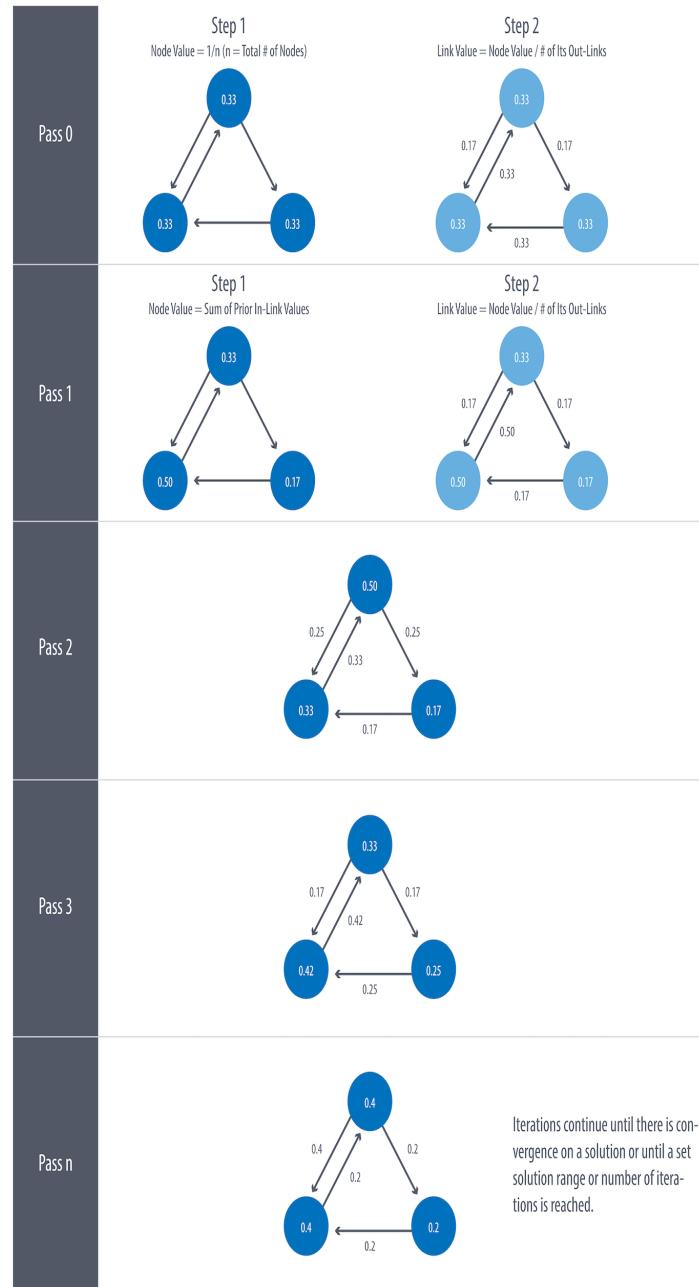


Figura 5-11. Cada iteración de PageRank tiene dos pasos de cálculo: uno para actualizar los valores de los nodos y otro para actualizar los valores de enlace.

## Iteración, surfistas aleatorios y sumideros de rango

PageRank es un algoritmo iterativo que se ejecuta hasta que las puntuaciones convergen o hasta que se alcanza un número determinado de iteraciones.

Conceptualmente, PageRank asume que hay un internauta que visita las páginas siguiendo los enlaces o utilizando una URL aleatoria. Un factor de amortiguamiento  $d$  define la probabilidad de que el próximo clic se realice a través de un enlace. Puedes considerarlo como la probabilidad de que un surfista se aburra y

cambie aleatoriamente a otra página. Una puntuación de PageRank representa la probabilidad de que una página sea visitada a través de un enlace entrante y no aleatoriamente.

Un nodo, o grupo de nodos, sin relaciones salientes (también llamado nodo colgante) puede monopolizar la puntuación de PageRank al negarse a compartir. Esto se conoce como un sumidero de rango. Se puede imaginar esto como un surfista que se atasca en una página, o un subconjunto de páginas, sin salida. Otra dificultad es creada por los nodos que se apuntan solo entre sí en un grupo. Las referencias circulares causan un aumento en sus filas a medida que el surfista rebota entre los nodos. Estas situaciones se describen en la [Figura 5-12](#).

### Rank Sinks Monopolize Rank Scores

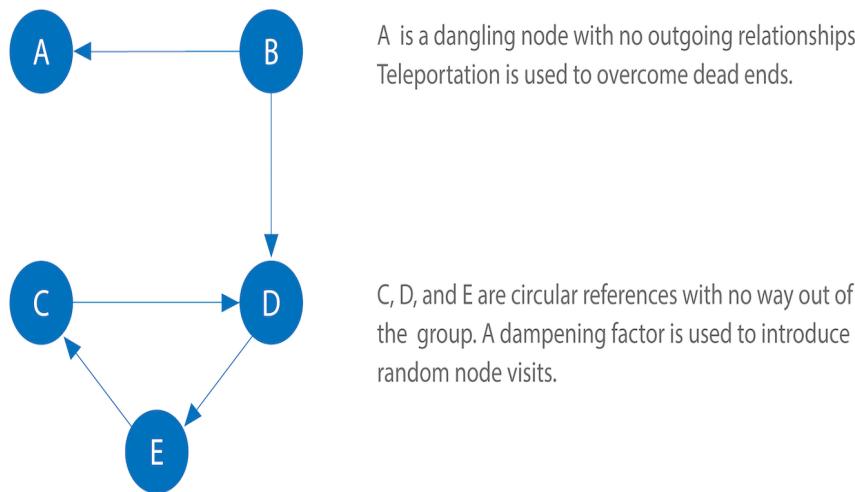


Figura 5-12. El sumidero de rango es causado por un nodo, o grupo de nodos, sin relaciones salientes.

Hay dos estrategias utilizadas para evitar los sumideros de rango. Primero, cuando se alcanza un nodo que no tiene relaciones salientes, PageRank asume relaciones salientes con todos los nodos. Atravesar estos enlaces invisibles a veces se llama teletransportación. En segundo lugar, el factor de amortiguación brinda otra oportunidad para evitar los sumideros al introducir una probabilidad de enlace directo frente a visitas de nodos aleatorios. Cuando establece  $d$  en 0.85, un nodo completamente aleatorio es visitado el 15% del tiempo.

Aunque la fórmula original recomienda un factor de atenuación de 0,85, su uso inicial fue en la World Wide Web con una distribución de enlaces de ley de poder (la mayoría de las páginas tienen muy pocos enlaces y unas pocas tienen muchos). La reducción del factor de amortiguación reduce la probabilidad de seguir rutas de larga relación antes de realizar un salto aleatorio. A su vez, esto aumenta la contribución de los predecesores inmediatos de un nodo a su puntaje y rango.

Si ve resultados inesperados de PageRank, vale la pena hacer un análisis exploratorio de la gráfica para ver si alguno de estos problemas es la causa. Lea el artículo de Ian Rogers, "[El algoritmo de PageRank de Google y cómo funciona](#)" para obtener más información.

## ¿Cuándo debo usar PageRank?

PageRank ahora se utiliza en muchos dominios fuera de la indexación web. Utilice este algoritmo siempre que busque una amplia influencia en una red. Por ejemplo, si está buscando apuntar a un gen que tenga el mayor impacto general en una función biológica, es posible que no sea el más conectado. De hecho, puede ser el gen con más relaciones con otras funciones más significativas.

Ejemplos de casos de uso incluyen:

- Presentar a los usuarios recomendaciones de otras cuentas que deseen seguir (Twitter utiliza PageRank personalizado para esto). El algoritmo se ejecuta en un gráfico que contiene intereses

compartidos y conexiones comunes. El enfoque se describe con más detalle en el documento "[WTF: El servicio de quién seguir en Twitter](#)" , por P. Gupta et al.

- Predicción del flujo de tráfico y movimiento humano en espacios públicos o calles. El algoritmo se ejecuta en un gráfico de intersecciones de carreteras, donde el puntaje de PageRank refleja la tendencia de las personas a estacionar o finalizar su viaje en cada calle. Esto se describe con más detalle en "[Caminos naturales autoorganizados para predecir el flujo de tráfico: un estudio de sensibilidad](#)" , un documento de B. Jiang, S. Zhao y J. Yin.
- Como parte de los sistemas de detección de anomalías y fraudes en las industrias de la salud y los seguros. PageRank ayuda a revelar a los médicos o proveedores que se están comportando de una manera inusual, y las calificaciones se incorporan a un algoritmo de aprendizaje automático.

David Gleich describe muchos más usos para el algoritmo en su artículo, "[PageRank Beyond the Web](#)" .

## PageRank con Apache Spark

Ahora estamos listos para ejecutar el algoritmo de PageRank. GraphFrames admite dos implementaciones de PageRank:

- La primera implementación ejecuta PageRank para un número fijo de iteraciones. Esto se puede ejecutar configurando el parámetro maxIter .
- La segunda implementación ejecuta PageRank hasta la convergencia. Esto se puede ejecutar configurando el parámetro tol .

### PageRank con un número fijo de iteraciones

Veamos un ejemplo del enfoque de iteraciones fijas:

```
resultados = g . pageRank ( resetProbability = 0.15 , maxIter = 20 ) resultados . vértices . sort ( "pagerank" , ascendente = Falso ) .  
mostrar ()
```

### PROPIA

Observe en Spark que el factor de amortiguación se denomina más intuitivamente la probabilidad de reinicio , con el valor inverso. En otras palabras, resetProbability = 0.15 en este ejemplo es equivalente a dampingFactor: 0.85 en Neo4j.

Si ejecutamos ese código en pyspark veremos esta salida:

#### carné de identidad rango de página

Doug	2.2865372087512252
marca	2.1424484186137263
Alicia	1.520330830262095
Miguel	0.7274429252585624
Bridget	0.7274429252585624
Charles	0.5213852310709753
Amy	0.5097143486157744
David	0.36655842368870073
James	0.1981396884803788

Como podríamos esperar, Doug tiene el PageRank más alto porque todos los demás usuarios lo siguen en su subgrafo. Aunque Mark solo tiene un seguidor, ese seguidor es Doug, por lo que Mark también se considera importante en este gráfico. No solo es importante la cantidad de seguidores, sino también la importancia de esos seguidores.

### PROPIA

Las relaciones en el gráfico en el que ejecutamos el algoritmo PageRank no tienen pesos, por lo que cada relación se considera igual. Las ponderaciones de las relaciones se agregan especificando una columna de ponderación en el DataFrame de las relaciones.

## PageRank hasta la convergencia

Y ahora probemos la implementación de convergencia que ejecutará PageRank hasta que se cierre en una solución dentro de la tolerancia establecida:

```
resultados = g . pageRank ( resetProbabilidad = 0.15 , tol = 0.01 ) resultados . vértices . sort ( "pagerank" , ascendente = Falso ) .
mostrar ()
```

Si ejecutamos ese código en pyspark veremos esta salida:

### carné de identidad rango de página

Doug	2.2233188859989745
marca	2.090451188336932
Alicia	1.5056291439101062
Miguel	0.733738785109624
Bridget	0.733738785109624
Amy	0.559446807245026
Charles	0.5338811076334145
David	0.40232326274180685
James	0.21747203391449021

Los puntajes de PageRank para cada persona son ligeramente diferentes a los de la variante de número de iteraciones fija, pero como es de esperar, su orden sigue siendo el mismo.

### PROPIA

Aunque la convergencia en una solución perfecta puede sonar ideal, en algunos escenarios, el PageRank no puede converger matemáticamente. Para gráficos más grandes, la ejecución de PageRank puede ser prohibitivamente larga. Un límite de tolerancia ayuda a establecer un rango aceptable para un resultado convergente, pero muchos optan por usar (o combinar este enfoque con) la opción de iteración máxima en su lugar. La configuración de iteración máxima generalmente proporcionará más consistencia de rendimiento. Independientemente de la opción que elija, es posible que deba probar varios límites diferentes para encontrar lo que funciona para su conjunto de datos. Las gráficas más grandes generalmente requieren más iteraciones o menor tolerancia que las gráficas de tamaño mediano para una mejor precisión.

## PageRank con Neo4j

También podemos ejecutar PageRank en Neo4j. Una llamada al siguiente procedimiento calculará el PageRank para cada uno de los nodos de nuestro gráfico:

```
CALL algo.pageRank.stream ( 'User' , 'FOLLOWS' , {iterations: 20, dampingFactor: 0.85} ) nodeId YIELD, puntuación RETURN
algo.getNodeById (nodeId). ID de la página AS , puntuación ORDEN POR puntuación DESC
```

Ejecutar este procedimiento da el siguiente resultado:

### página Puntuación

Doug	1.6704119999999998
marca	1.5610085
Alicia	1.1106700000000003
Bridget	0.535373
Miguel	0.535373
Amy	0.385875
Charles	0.3844895
David	0.2775
James	0.1500000000000002

Al igual que con el ejemplo de Spark, Doug es el usuario más influyente, y Mark lo sigue de cerca como el único usuario que Doug sigue. Podemos ver la importancia de los nodos en relación unos con otros en la [Figura 5-13](#).

### NOTA

Las implementaciones de PageRank varían, por lo que pueden producir una puntuación diferente incluso cuando el orden es el mismo. Neo4j inicializa los nodos con un valor de 1 menos el factor de amortiguación, mientras que Spark usa un valor de 1. En este caso, las clasificaciones relativas (el objetivo de PageRank) son idénticas, pero los valores de puntuación subyacentes utilizados para alcanzar esos resultados son diferentes.

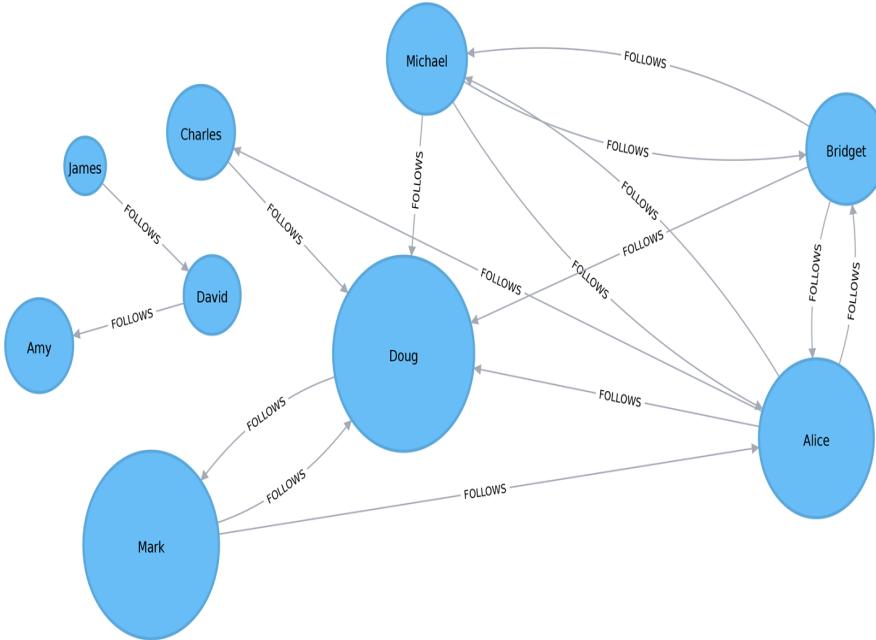


Figura 5-13. Visualización de PageRank

### PROPIA

Al igual que con nuestro ejemplo de Spark, las relaciones en el gráfico en el que ejecutamos el algoritmo PageRank no tienen ponderaciones, por lo que cada relación se considera igual. Se pueden considerar los pesos de la relación incluyendo la propiedad `weightProperty` en la configuración pasada al procedimiento de PageRank. Por ejemplo, si las relaciones tienen un peso de propiedad que contiene pesos, pasaríamos la siguiente configuración al procedimiento: `weightProperty: "weight"`.

## Variación del PageRank: PageRank personalizado

El PageRank personalizado (PPR) es una variante del algoritmo de PageRank que calcula la importancia de los nodos en un gráfico desde la perspectiva de un nodo específico. Para PPR, los saltos aleatorios se refieren a un conjunto dado de nodos de inicio. Esto sesga los resultados hacia, o se personaliza para, el nodo de inicio. Este sesgo y la localización hacen que la PPR sea útil para recomendaciones altamente específicas.

### PageRank personalizado con Apache Spark

Podemos calcular el puntaje de PageRank personalizado para un nodo dado pasando el parámetro `sourceId`. El siguiente código calcula el PPR para Doug:

```

me = "Doug"
resultados = g . pageRank ( resetProbability = 0.15 , maxIter = 20 , sourceId = me ) . people_to_follow = resultados . vértices .
sort ( "pagerank" , ascendente = Falso ) . already_follows = list ( g . bordes . filtro ( f "src = '{me}'" ) . toPandas () [ "dst" ] )
people_to_exclude = already_follows + [ me ] . people_to_follow [ ~ people_to_follow . ID . ISIN ( people_to_exclude ) ] . mostrar ()
  
```

Los resultados de esta consulta podrían usarse para hacer recomendaciones para las personas que Doug debería seguir. Tenga en cuenta que también nos aseguramos de excluir a las personas que Doug ya sigue, así como a sí mismo, de nuestro resultado final.

Si ejecutamos ese código en pyspark veremos esta salida:

**carné de identidad rango de página**

Alicia

0.1650183746272782

Miguel	0.048842467744891996
Bridget	0.048842467744891996
Charles	0.03497796119878669
David	0.0
James	0.0
Amy	0.0

Alice es la mejor sugerencia para alguien que Doug debería seguir, pero también podemos sugerirle a Michael y Bridget.

## Resumen

Los algoritmos de centralidad son una herramienta excelente para identificar personas influyentes en una red. En este capítulo hemos aprendido acerca de los algoritmos prototípicos de centralidad: Centralidad de grados, Centralidad de proximidad, Centralidad de intermediación y PageRank. También hemos cubierto varias variaciones para tratar temas como los tiempos de ejecución prolongados y los componentes aislados, así como las opciones para usos alternativos.

Hay muchos usos amplios para los algoritmos de centralidad, y fomentamos su exploración para una variedad de análisis. Puede aplicar lo que hemos aprendido para ubicar los puntos de contacto óptimos para difundir información, encontrar a los intermediarios ocultos que controlan el flujo de recursos y descubrir a los jugadores de poder indirectos que acechan en las sombras.

A continuación, pasaremos a los algoritmos de detección de la comunidad que analizan grupos y particiones.

# Capítulo 6. Algoritmos de detección de la comunidad.

---

La formación de la comunidad es común en todos los tipos de redes, y su identificación es esencial para evaluar el comportamiento del grupo y los fenómenos emergentes. El principio general para encontrar comunidades es que sus miembros tendrán más relaciones dentro del grupo que con nodos fuera de su grupo. La identificación de estos conjuntos relacionados revela grupos de nodos, grupos aislados y estructura de red. Esta información ayuda a inferir comportamientos o preferencias similares de grupos de pares, estimar la capacidad de recuperación, encontrar relaciones anidadas y preparar datos para otros análisis. Los algoritmos de detección de la comunidad también se usan comúnmente para producir la visualización de la red para la inspección general.

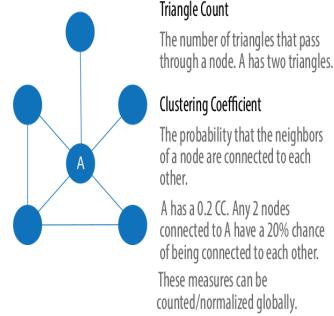
Proporcionaremos detalles sobre los algoritmos de detección de comunidad más representativos:

- Recuento de triángulos y coeficiente de agrupamiento para la densidad de relaciones en general
- Componentes fuertemente conectados y componentes conectados para encontrar clústeres conectados
- Propagación de etiquetas para inferir rápidamente grupos basados en etiquetas de nodos
- Modularidad de Louvain para observar la calidad y las jerarquías de agrupación

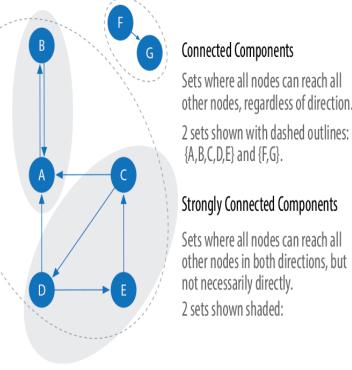
Explicaremos cómo funcionan los algoritmos y mostraremos ejemplos en Apache Spark y Neo4j. En los casos en que un algoritmo solo esté disponible en una plataforma, le proporcionaremos un solo ejemplo. Usamos relaciones ponderadas para estos algoritmos porque normalmente se usan para capturar el significado de diferentes relaciones.

[La Figura 6-1](#) ofrece una descripción general de las diferencias entre los algoritmos de detección de la comunidad que se tratan aquí, y la [Tabla 6-1](#) proporciona una referencia rápida sobre lo que calcula cada algoritmo con usos de ejemplo.

## Measuring Algorithms

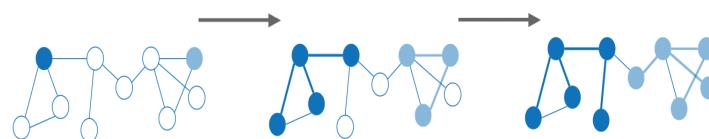


## Components Algorithms



## Label Propagation Algorithm

Spread labels to or from neighbors to find clusters.

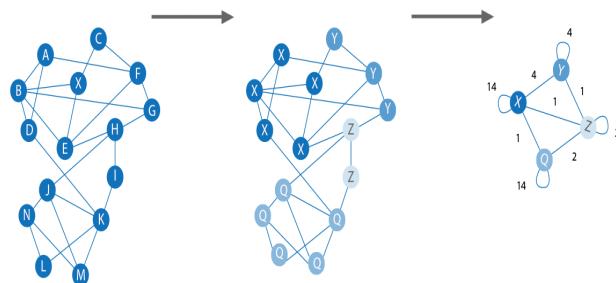


Run over multiple iterations.

Weights of relationships and/or nodes are often used to determine label "popularity" in a group.

## Louvain Modularity Algorithm

Find clusters by moving nodes into higher relationship density groups and aggregating into supercommunities.



Run over multiple iterations.

Relationship weights and totals are used to determine grouping.

Figura 6-1. Algoritmos de detección representativos de la comunidad

## NOTA

Utilizamos los términos conjunto , partición , clúster , grupo y comunidad de manera intercambiable. Estos términos son diferentes formas de indicar que se pueden agrupar nodos similares. Los algoritmos de detección de la comunidad también se denominan algoritmos de agrupamiento y partición. En cada sección, utilizamos los términos más destacados en la literatura para un algoritmo en particular.

Tabla 6-1. Visión general de los algoritmos de detección de la comunidad

Tipo de algoritmo	Que hace	Ejemplo de uso	Ejemplo de chispa	Ejemplo neo4j
<a href="#"><u>Conteo de triángulos y coeficiente de agrupamiento</u></a>	Mide cuántos nodos forman triángulos y el grado en que los nodos tienden a agruparse	Estimación de la estabilidad del grupo y si la red puede exhibir comportamientos de "mundo pequeño" vistos en gráficos con grupos muy unidos.	Sí	Sí
<a href="#"><u>Componentes fuertemente conectados</u></a>	Busca grupos donde se puede acceder a cada nodo desde cualquier otro nodo en ese mismo grupo siguiendo la dirección de las relaciones	Hacer recomendaciones de productos basadas en afiliación grupal o artículos similares.	Sí	Sí
<a href="#"><u>Componentes conectados</u></a>	Busca grupos donde se puede acceder a cada nodo desde cualquier otro nodo en ese mismo grupo, independientemente de la dirección de las relaciones	Realiza agrupaciones rápidas para otros algoritmos e identifica islas.	Sí	Sí

<a href="#"><u>Propagación de etiquetas</u></a>	Infiere los clusters al difundir etiquetas basadas en las mayorías del vecindario	Comprender el consenso en las comunidades sociales o encontrar combinaciones peligrosas de posibles medicamentos prescritos conjuntamente	Sí	Sí
<a href="#"><u>Modularidad de Lovaina</u></a>	Maximiza la presunta precisión de las agrupaciones al comparar los pesos y densidades de las relaciones con una estimación o promedio definidos	En el análisis de fraude, evaluar si un grupo tiene solo unos pocos malos comportamientos discretos o está actuando como una red de fraude	No	Sí

En primer lugar, describiremos los datos de nuestros ejemplos y seguiremos importando los datos a Spark y Neo4j. Los algoritmos se cubren en el orden listado en la [Tabla 6-1](#). Para cada uno, encontrará una breve descripción y consejos sobre cuándo usarlo. La mayoría de las secciones también incluyen una guía sobre cuándo usar algoritmos relacionados. Demostramos código de ejemplo usando datos de muestra al final de cada sección de algoritmo.

## PROPINA

Cuando use algoritmos de detección de la comunidad, sea consciente de la densidad de las relaciones.

Si la gráfica es muy densa, puede terminar con todos los nodos congregándose en uno o solo unos pocos grupos. Puede contrarrestar esto al filtrar por grado, pesos de relación o métricas de similitud.

Por otro lado, si el gráfico es muy escaso con pocos nodos conectados, puede terminar con cada nodo en su propio grupo. En este caso, intente incorporar tipos de relaciones adicionales que contengan información más relevante.

## Ejemplo de datos de gráfico: el gráfico de dependencia de software

Los gráficos de dependencia son particularmente adecuados para demostrar las diferencias a veces sutiles entre los algoritmos de detección de la comunidad porque tienden a estar más conectados y jerárquicos. Los ejemplos en este capítulo se ejecutan en un gráfico que contiene dependencias entre bibliotecas de Python, aunque los gráficos de dependencia se utilizan en varios campos, desde el software hasta las redes de energía. Este tipo de gráfico de dependencia del software es utilizado por los desarrolladores para realizar un seguimiento de las interdependencias transitivas y los conflictos en los proyectos de software. Puede descargar los nodos y los archivos del [repositorio GitHub del libro](#).

Tabla 6-2. sw-nodes.csv

### carné de identidad

seis
pandas
adormecido
python-dateutil
pytz
pyspark
matplotlib
espacioso
py4j
jupyter
jpy-console
nbconvert
ipykernel
cliente jpy
jpy-core

Tabla 6-3. sw-relationships.csv

src	dst	relación
pandas	adormecido	DEPENDE DE
pandas	pytz	DEPENDE DE
pandas	python-dateutil	DEPENDE DE
python-dateutil	seis	DEPENDE DE
pyspark	py4j	DEPENDE DE
matplotlib	adormecido	DEPENDE DE
matplotlib	python-dateutil	DEPENDE DE
matplotlib	seis	DEPENDE DE
matplotlib	pytz	DEPENDE DE
espacioso	seis	DEPENDE DE
espacioso	adormecido	DEPENDE DE
jupyter	nbconvert	DEPENDE DE
jupyter	ipykernel	DEPENDE DE
jupyter	jpy-console	DEPENDE DE
jpy-console	cliente jpy	DEPENDE DE
jpy-console	ipykernel	DEPENDE DE
cliente jpy	jpy-core	DEPENDE DE
nbconvert	jpy-core	DEPENDE DE

[La figura 6-2](#) muestra la gráfica que queremos construir. Mirando este gráfico, vemos que hay tres grupos de bibliotecas. Podemos usar visualizaciones en conjuntos de datos más pequeños como una herramienta para ayudar a validar los grupos derivados de los algoritmos de detección de la comunidad.

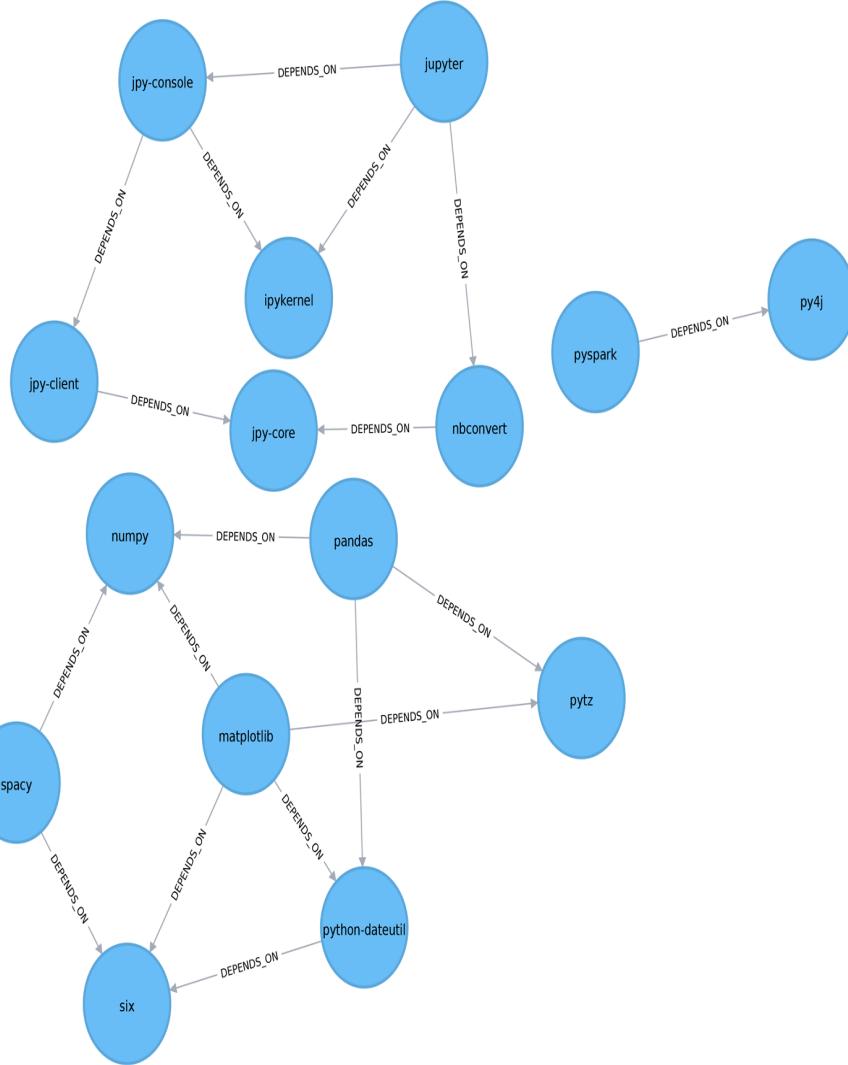


Figura 6-2. El modelo grafico

Vamos a crear gráficos en Spark y Neo4j a partir de los archivos CSV de ejemplo.

## Importando los datos en Apache Spark

Primero importaremos los paquetes que necesitamos de Apache Spark y el paquete GraphFrames:

desde los gráficos de importación \*

La siguiente función crea un GraphFrame a partir de los archivos CSV de ejemplo:

```
def create_software_graph (): nodes = spark . lea . csv ( "data / sw-nodes.csv" , header = True ) relationships = spark . lea . csv ( "data / sw-relationships.csv" , header = True ) devuelve GraphFrame ( nodos , relaciones )
```

Ahora llamemos a esa función:

```
g = create_software_graph ()
```

## Importando los datos en Neo4j

A continuación haremos lo mismo para Neo4j. La siguiente consulta importa los nodos:

```
CON "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base WITH base + "sw-nodes.csv" COMO uri CSV DE
CARGA CON LOS JEFATORES DE uri AS fila MERGE (: Biblioteca { id : row. Id })
```

Y esto importa las relaciones:

```
CON "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base WITH base + "sw-relationships.csv" AS uri CSV DE
CARGA CON LOS JEFATORES DE uri AS fila MATCH (fuente : Biblioteca { id : row.src }) MATCH (destino: Biblioteca { id :
row.dst }) MERGE (fuente) - [: DEPENDS_ON] -> (destino)
```

¡Ahora que tenemos nuestros gráficos cargados, está en los algoritmos!

## Conteo de triángulos y coeficiente de agrupamiento

Los algoritmos de Conteo de triángulos y Coeficiente de agrupamiento se presentan juntos porque a menudo se usan juntos. El recuento de triángulos determina el número de triángulos que pasan a través de cada nodo en el gráfico. Un triángulo es un conjunto de tres nodos, donde cada nodo tiene una relación con todos los demás nodos. Triangle Count también se puede ejecutar globalmente para evaluar nuestro conjunto de datos general.

### NOTA

Las redes con un alto número de triángulos tienen más probabilidades de exhibir estructuras y comportamientos del mundo pequeño.

El objetivo del algoritmo del coeficiente de agrupación es medir qué tan bien está agrupado un grupo en comparación con qué tan bien podría estar agrupado. El algoritmo utiliza el recuento de triángulos en sus cálculos, que proporciona una relación de triángulos existentes a las posibles relaciones. Un valor máximo de 1 indica una camarilla donde cada nodo está conectado a cada otro nodo.

Hay dos tipos de coeficientes de agrupamiento: agrupamiento local y agrupamiento global.

## Coeficiente de agrupamiento local

El coeficiente de agrupamiento local de un nodo es la probabilidad de que sus vecinos también estén conectados. El cálculo de esta puntuación implica el conteo de triángulos.

El coeficiente de agrupamiento de un nodo se puede encontrar al multiplicar el número de triángulos que pasan por el nodo por dos y luego dividirlo por el número máximo de relaciones en el grupo, que es siempre el grado de ese nodo, menos uno. En la [Figura 6-3](#) se muestran ejemplos de triángulos y coeficientes de agrupamiento diferentes para un nodo con cinco relaciones .

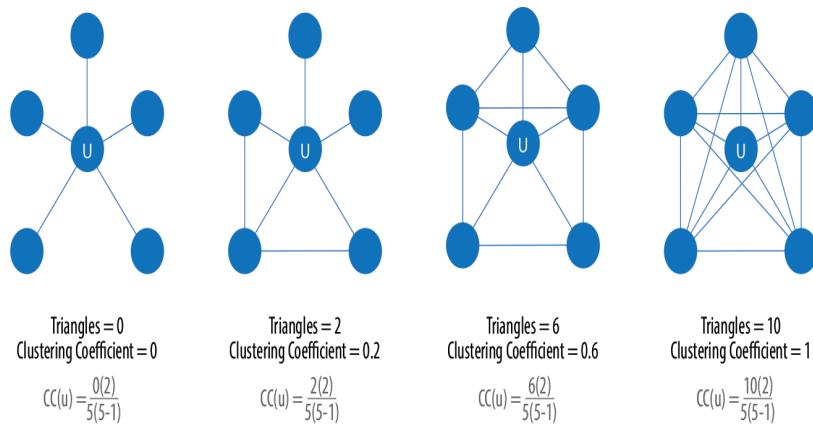


Figura 6-3. Recuento de triángulos y coeficientes de agrupamiento para el nodo u

Tenga en cuenta que en la [Figura 6-3](#) , usamos un nodo con cinco relaciones que hace que parezca que el coeficiente de agrupación siempre equivaldrá al 10% del número de triángulos. Podemos ver que este no es el caso cuando modificamos el número de relaciones. Si cambiamos el segundo ejemplo para que tenga cuatro relaciones (y los mismos dos triángulos), entonces el coeficiente es 0.33.

El coeficiente de agrupamiento para un nodo usa la fórmula:

$$CC(u) = \frac{2R_u}{K_u(K_u - 1)}$$

dónde:

- U es un nodo.
- R (u) es el número de relaciones a través de los vecinos de u (esto puede obtenerse utilizando el número de triángulos que pasan a través de u ).
- K (u) es el grado de u .

## Coeficiente de agrupamiento global

El coeficiente de agrupamiento global es la suma normalizada de los coeficientes de agrupamiento local.

Los coeficientes de agrupación nos brindan un medio eficaz para encontrar grupos obvios como camarillas, donde cada nodo tiene una relación con todos los demás nodos, pero también podemos especificar umbrales para establecer niveles (por ejemplo, donde los nodos están conectados en un 40%).

## ¿Cuándo debo usar el recuento de triángulos y el coeficiente de agrupamiento?

Use el recuento de triángulos cuando necesite determinar la estabilidad de un grupo o como parte del cálculo de otras medidas de red, como el coeficiente de agrupamiento. El conteo de triángulos es popular en el análisis de redes sociales, donde se usa para detectar comunidades.

El coeficiente de agrupación puede proporcionar la probabilidad de que los nodos seleccionados al azar se conecten. También puede usarlo para evaluar rápidamente la cohesión de un grupo específico o su red general. Juntos, estos algoritmos se utilizan para estimar la resistencia y buscar estructuras de red.

Ejemplos de casos de uso incluyen:

- Identificación de características para clasificar un sitio web determinado como contenido de spam.  
Esto se describe en "[Algoritmos de transmisión semiautomática eficientes para el conteo de triángulos locales en gráficos masivos](#)" , un artículo de L. Becchetti et al.
- Investigando la estructura comunitaria del gráfico social de Facebook, donde los investigadores encontraron vecindarios densos de usuarios en un gráfico global por lo demás disperso. Encuentre este estudio en el documento "[La anatomía de la gráfica social de Facebook](#)" , por J. Ugander et al.
- Exploración de la estructura temática de la web y detección de comunidades de páginas con temas comunes basados en los vínculos recíprocos entre ellos. Para obtener más información, consulte "[La curvatura de co-enlaces descubre capas temáticas ocultas en la World Wide Web](#)" , por J.-P. Eckmann y E. Moses.

## Recuento de triángulos con chispa de apache

Ahora estamos listos para ejecutar el algoritmo de conteo de triángulos. Podemos usar el siguiente código para hacer esto:

```
resultado = g . triangleCount () ( resultado . sort ( "count" , ascendeante = False ) . filter ( 'count > 0' ) . show () )
```

Si ejecutamos ese código en pyspark veremos esta salida:

### contar carné de identidad

1	jupyter
1	python-dateutil
1	seis
1	ipykernel
1	matplotlib
1	jpy-console

Un triángulo en este gráfico indicaría que dos de los vecinos de un nodo también son vecinos. Seis de nuestras bibliotecas participan en tales triángulos.

¿Qué pasa si queremos saber qué nodos están en esos triángulos? Ahí es donde entra una corriente de triángulos . Para esto, necesitamos Neo4j.

## Triángulos con Neo4j

Obtener un flujo de los triángulos no está disponible usando Spark, pero podemos devolverlo usando Neo4j:

```
CALL algo.triangle.stream ( "Library" , "DEPENDS_ON" ) YIELD nodeA, nodeB, nodeC RETURN algo.getNodeById (nodeA).id AS nodeA, algo.getNodeById (nodeB).id AS nodeB, algo.getNodeById (nodeC).id AS nodeC
```

Ejecutar este procedimiento da el siguiente resultado:

nodoA	nodoB	nodoC
matplotlib	seis	python-dateutil
jupyter	jpy-console	ipykernel

Vemos las mismas seis bibliotecas como lo hicimos antes, pero ahora sabemos cómo están conectadas. matplotlib, six y python-dateutil forman un triángulo. jupyter, jpy-console y ipykernel forman el otro.

Podemos ver estos triángulos visualmente en la [Figura 6-4](#).

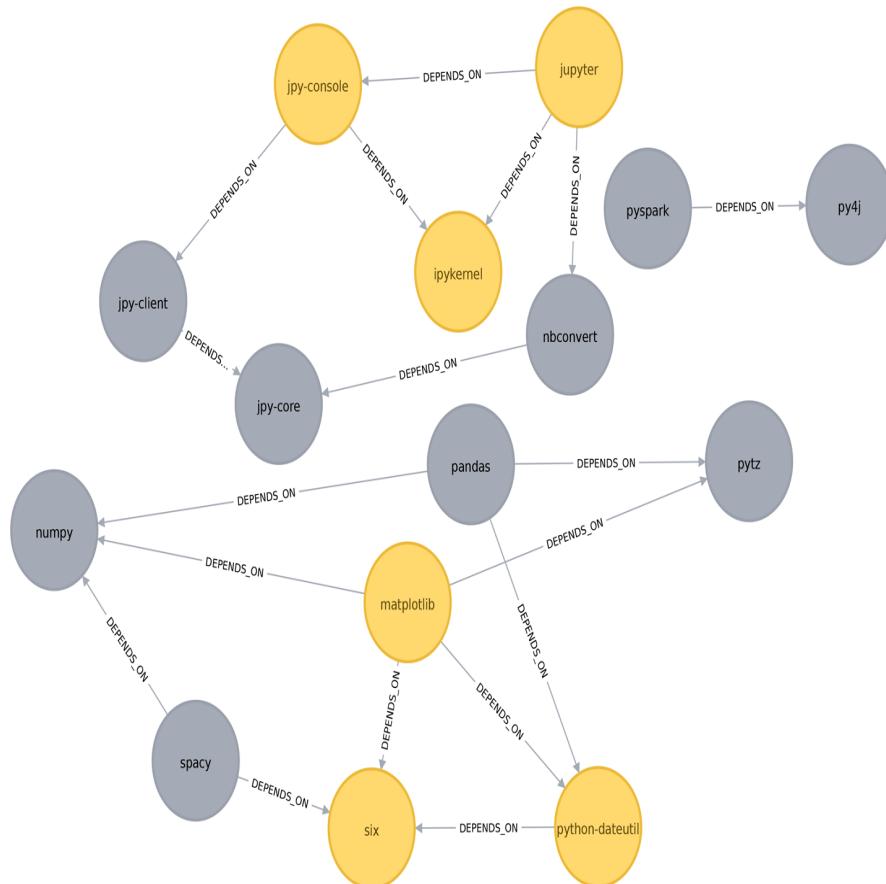


Figura 6-4. Triángulos en el gráfico de dependencia del software.

## Coeficiente de agrupamiento local con Neo4j

También podemos calcular el coeficiente de agrupamiento local. La siguiente consulta calculará esto para cada nodo:

```
CALL algo.triangleCount.stream ( 'Library' , 'DEPENDS_ON' ) YIELD nodeId, triángulos, coeficiente DONDE coeficiente > 0
RETURN algo.getNodeById (nodeId).ID de la biblioteca de AS , coeficiente ORDEN DE CODIGO DESC
```

Ejecutar este procedimiento da el siguiente resultado:

biblioteca	coeficiente
ipykernel	1.0
jupyter	0.3333333333333333
jpy-console	0.3333333333333333
seis	0.3333333333333333

python-dateutil 0.3333333333333333

matplotlib 0.1666666666666666

ipykernel tiene una puntuación de 1, lo que significa que todos los vecinos de ipykernel son vecinos entre sí. Podemos ver claramente eso en la [Figura 6-4](#). Esto nos dice que la comunidad directamente alrededor de ipykernel es muy cohesiva.

Hemos filtrado los nodos con una puntuación del coeficiente de 0 en este ejemplo de código, pero los nodos con coeficientes bajos también pueden ser interesantes. Una puntuación baja puede ser un indicador de que un nodo es un [agujero estructural](#), un nodo que está bien conectado a los nodos en diferentes comunidades que no están conectadas entre sí. Este es un método para encontrar puentes potenciales que analizamos en el [Capítulo 5](#).

## Componentes fuertemente conectados

El algoritmo de componentes fuertemente conectados (SCC) es uno de los primeros algoritmos de grafos. SCC encuentra conjuntos de nodos conectados en un gráfico dirigido donde cada nodo es accesible en ambas direcciones desde cualquier otro nodo en el mismo conjunto. Sus operaciones de tiempo de ejecución se escalan bien, de forma proporcional al número de nodos. En la [Figura 6-5](#) puede ver que los nodos en un grupo SCC no necesitan ser vecinos inmediatos, pero debe haber rutas direccionales entre todos los nodos en el conjunto.

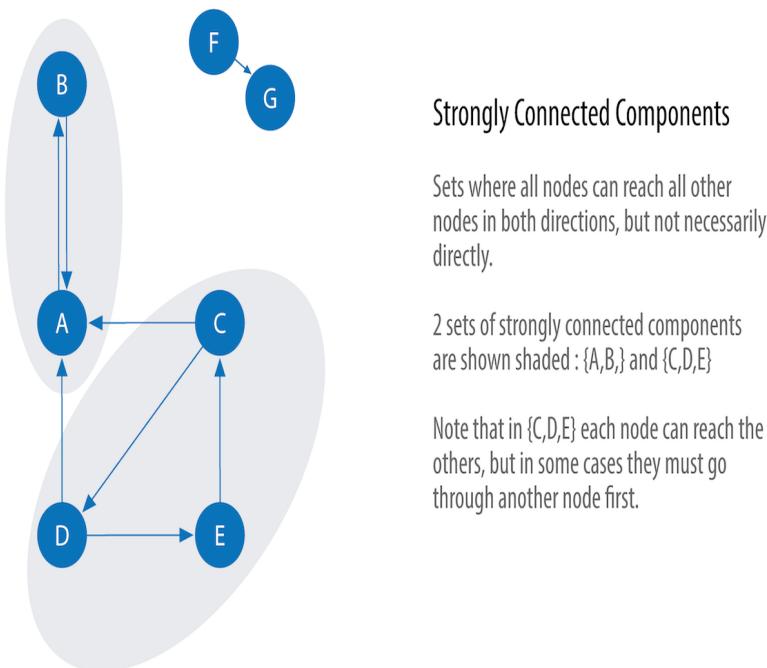


Figura 6-5. Componentes fuertemente conectados

### NOTA

La descomposición de un gráfico dirigido en sus componentes fuertemente conectados es una aplicación clásica del [algoritmo de búsqueda en profundidad](#). Neo4j usa DFS bajo el capó como parte de su implementación del algoritmo SCC.

## ¿Cuándo debo usar componentes fuertemente conectados?

Use los componentes fuertemente conectados como un paso temprano en el análisis de gráficos para ver cómo está estructurada una gráfica o para identificar agrupaciones cerradas que puedan justificar una investigación independiente. Un componente que está fuertemente conectado se puede usar para perfilar comportamientos similares o inclinaciones en un grupo para aplicaciones como motores de recomendación.

Muchos algoritmos de detección de comunidades, como SCC, se utilizan para encontrar y colapsar agrupamientos en nodos individuales para un análisis de interclúster adicional. También puede usar SCC para

visualizar los ciclos para análisis como encontrar procesos que podrían interrumpirse porque cada subprocesso está esperando a que otro miembro actúe.

Ejemplos de casos de uso incluyen:

- Encontrar el conjunto de empresas en las que cada miembro, directa o indirectamente, posee acciones de todos los demás miembros, como en "["La red de control corporativo global"](#)" , un análisis de las poderosas empresas transnacionales por S. Vitali, JB Glattfelder y S. Battiston.
- Calcular la conectividad de diferentes configuraciones de red al medir el rendimiento de enrutamiento en redes inalámbricas multihop. Lea más en "["Rendimiento de enrutamiento en presencia de enlaces unidireccionales en redes inalámbricas Multihop"](#)" , por MK Marina y SR Das.
- Actuar como el primer paso en muchos algoritmos de gráficos que funcionan solo en gráficos fuertemente conectados. En las redes sociales encontramos muchos grupos fuertemente conectados. En estos conjuntos, las personas a menudo tienen preferencias similares, y el algoritmo SCC se usa para encontrar dichos grupos y sugerir páginas para "Me gusta" o productos para comprar a las personas del grupo que aún no lo han hecho.

### **PROPIA**

Algunos algoritmos tienen estrategias para escapar de bucles infinitos, pero si escribimos nuestros propios algoritmos o encontramos procesos que no terminan, podemos usar SCC para verificar los ciclos.

## **Componentes fuertemente conectados con Apache Spark**

Comenzando con Apache Spark, primero importaremos los paquetes que necesitamos de Spark y el paquete de GraphFrames:

```
de graphframes importar * desde pyspark.sql importación funciones como F
```

Ahora estamos listos para ejecutar el algoritmo de componentes fuertemente conectados. Lo usaremos para determinar si hay dependencias circulares en nuestro gráfico.

### **NOTA**

Dos nodos solo pueden estar en el mismo componente fuertemente conectado si hay caminos entre ellos en ambas direcciones.

Escribimos el siguiente código para hacer esto:

```
resultado = g . stronglyConnectedComponents ( maxiter = 10 ) ( resultado . especie ( "componente" ) . GroupBy ( "componente" ) . agg ( F . collect_list ( "id" ) . alias ( "bibliotecas" ) ) . espectáculo ( truncar = Falso ) )
```

Si ejecutamos ese código en pyspark veremos esta salida:

<b>componente</b>	<b>bibliotecas</b>
180388626432	[jpy-core]
223338299392	[spacy]
498216206336	[numpy]
523986010112	[seis]
549755813888	[pandas]
558345748480	[nbconvert]
661424963584	[ipykernel]
721554505728	[jupyter]
764504178688	[jpy-cliente]
833223655424	[pytz]
910533066752	[python-dateutil]
936302870528	[pyspark]

---

 944892805120 [matplotlib]
 

---

 1099511627776 [jpy-console]
 

---

 1279900254208 [py4j]
 

---

Es posible que observe que cada nodo de la biblioteca está asignado a un componente único. Esta es la partición o subgrupo al que pertenece, y como esperábamos (con suerte), cada nodo está en su propia partición. Esto significa que nuestro proyecto de software no tiene dependencias circulares entre estas bibliotecas.

## Componentes fuertemente conectados con Neo4j

Vamos a ejecutar el mismo algoritmo usando Neo4j. Ejecute la siguiente consulta para ejecutar el algoritmo:

```
CALL algo.secc.stream ( "Library" , "DEPENDS_ON" ) YIELD nodeId, partición RETURN , recopile (algo.getNodeById (nodeId))
Bibliotecas AS ORDER BY tamaño (bibliotecas) DESC
```

Los parámetros pasados a este algoritmo son:

**Biblioteca**

La etiqueta de nodo para cargar desde el gráfico.

**DEPENDE DE**

El tipo de relación a cargar desde el gráfico.

Esta es la salida que veremos cuando ejecutemos la consulta:

### dividir bibliotecas

---

8	[ipykernel]
11	[seis]
2	[matplotlib]
5	[jupyter]
14	[python-dateutil]
13	[numpy]
4	[py4j]
7	[nbconvert]
1	[pyspark]
10	[jpy-core]
9	[jpy-cliente]
3	[spacy]
12	[pandas]
6	[jpy-console]
0	[pytz]

---

Al igual que con el ejemplo de Spark, cada nodo está en su propia partición.

Hasta ahora, el algoritmo solo ha revelado que nuestras bibliotecas de Python se comportan muy bien, pero creemos una dependencia circular en el gráfico para hacer las cosas más interesantes. Esto debería significar que terminaremos con algunos nodos en la misma partición.

La siguiente consulta agrega una biblioteca adicional que crea una dependencia circular entre py4j y pyspark:

```
MATCH (py4j: Library { id : "py4j" }) MATCH (pyspark: Library { id : "pyspark" }) MERGE (extra: Library { id : "extra" }) MERGE
(py4j) - [:DEPENDS_ON] -> (extra) MERGE (extra) - [:DEPENDS_ON] -> (pyspark)
```

Podemos ver claramente la dependencia circular que se creó en la [Figura 6-6](#) .

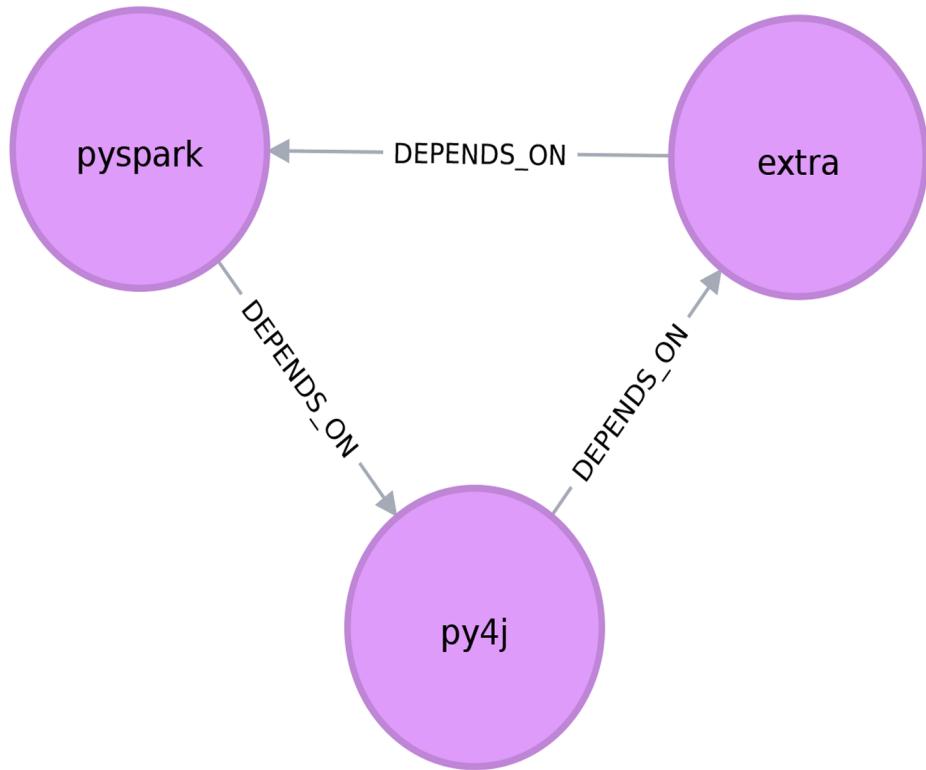


Figura 6-6. Una dependencia circular entre pyspark, py4j y extra

Ahora, si volvemos a ejecutar el algoritmo SCC, veremos un resultado ligeramente diferente:

#### **dividir bibliotecas**

1	[pyspark, py4j, extra]
8	[ipykernel]
11	[seis]
2	[matplotlib]
5	[jupyter]
14	[numpy]
13	[pandas]
7	[nbconvert]
10	[jpy-core]
9	[jpy-cliente]
3	[spacy]
15	[python-dateutil]
6	[jpy-console]
0	[pytz]

pyspark, py4j y extra son parte de la misma partición, ¡y los SCC nos ayudaron a encontrar la dependencia circular!

Antes de pasar al siguiente algoritmo, eliminaremos la biblioteca adicional y sus relaciones del gráfico.:

```
MATCH (extra: Library { id : "extra" }) DETACH DELETE extra
```

## Componentes conectados

El algoritmo de Componentes Conectados (a veces llamado Unión Buscar o Componentes Conectados Débilmente) encuentra conjuntos de nodos conectados en un gráfico no dirigido donde cada nodo es accesible desde cualquier otro nodo en el mismo conjunto. Se diferencia del algoritmo SCC porque solo necesita una ruta para existir entre pares de nodos en una dirección, mientras que SCC necesita una ruta para existir en ambas direcciones. Bernard A. Galler y Michael J. Fischer describieron por primera vez este algoritmo en su artículo de 1964, "[Un algoritmo de equivalencia mejorado](#)" .

## ¿Cuándo debo usar los componentes conectados?

Al igual que con SCC, los componentes conectados a menudo se usan al principio de un análisis para comprender la estructura de un gráfico. Debido a que se escala de manera eficiente, considere este algoritmo para gráficos que requieren actualizaciones frecuentes. Puede mostrar rápidamente nuevos nodos en común entre grupos, lo que es útil para el análisis, como la detección de fraudes.

Convierta en un hábito ejecutar componentes conectados para comprobar si un gráfico está conectado como un paso preparatorio para el análisis general de gráficos. Realizar esta prueba rápida puede evitar la ejecución accidental de algoritmos en solo un componente desconectado de un gráfico y obtener resultados incorrectos.

Ejemplos de casos de uso incluyen:

- Realizar un seguimiento de los grupos de registros de la base de datos, como parte del proceso de deduplicación. La deduplicación es una tarea importante en las aplicaciones de administración de datos maestros; El enfoque se describe con más detalle en "[Un algoritmo independiente de dominio eficiente para detectar registros de base de datos aproximadamente duplicados](#)" , por A. Monge y C. Elkan.
- Análisis de redes de citas. Un estudio utiliza Componentes Conectados para determinar qué tan bien conectada está una red, y luego para ver si la conectividad permanece si los nodos "concentrador" o "autoridad" se mueven del gráfico. Este caso de uso se explica más detalladamente en "[Gráfica de citas y caracterización de la literatura de informática](#)" , un artículo de Y. An, JCM Janssen y EE Milios.

## Componentes conectados con Apache Spark

Comenzando con Apache Spark, primero importaremos los paquetes que necesitamos de Spark y el paquete de GraphFrames:

**desde pyspark.sql funciones de importación como F**

Ahora estamos listos para ejecutar el algoritmo de Componentes Conectados.

### NOTA

Dos nodos pueden estar en el mismo componente conectado si hay una ruta entre ellos en cualquier dirección.

Escribimos el siguiente código para hacer esto:

```
resultado = g . connectedComponents () ( resultado . especie ( "componente" ) . GroupBy ( "componente" ) . agg ( F . collect_list ( "id" ) . alias ( "bibliotecas" ) ) . espectáculo ( truncar = False ))
```

Si ejecutamos ese código en pyspark veremos esta salida:

---

**componente bibliotecas**

---

180388626432 [jpy-core, nbconvert, ipykernel, jupyter, jpy-client, jpy-console]

---

223338299392 [spacy, numpy, six, pandas, pytz, python-dateutil, matplotlib]

---

936302870528 [pyspark, py4j]

---

Los resultados muestran tres grupos de nodos, que también se pueden ver en la [Figura 6-7](#) .

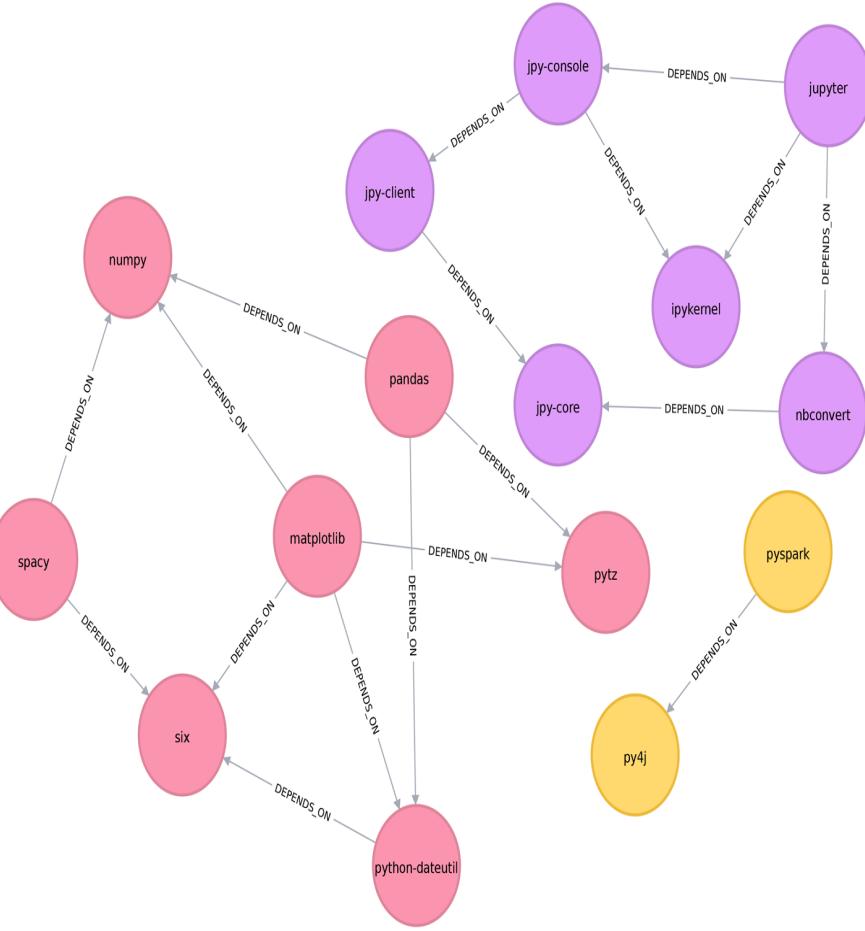


Figura 6-7. Grupos encontrados por el algoritmo de componentes conectados

En este ejemplo, es muy fácil ver que hay tres componentes solo por inspección visual. Este algoritmo muestra su valor más en gráficos más grandes, donde la inspección visual no es posible o requiere mucho tiempo.

## Componentes conectados con Neo4j

También podemos ejecutar este algoritmo en Neo4j ejecutando la siguiente consulta:

```
CALL algo.unionFind.stream( "Library" , "DEPENDS_ON" ) YIELD nodeId, setId RETURN setId, collect(algo.getNodeById(nodeId)) AS bibliotecas ORDER BY tamaño(bibliotecas) DESC
```

Los parámetros pasados a este algoritmo son:

Biblioteca

La etiqueta de nodo para cargar desde el gráfico.

DEPENDE DE

El tipo de relación a cargar desde el gráfico.

Aquí está la salida:

---

**Pon la identificación bibliotecas**

---

2 [Pytz, matplotlib, spacy, six, pandas, numpy, python-dateutil]

5 [jupyter, jpy-console, nbconvert, ipykernel, jpy-client, jpy-core]

1 [pyspark, py4j]

---

Como se esperaba, obtenemos exactamente los mismos resultados que obtuvimos con Spark.

Los dos algoritmos de detección de la comunidad que hemos cubierto hasta ahora son deterministas: devuelven los mismos resultados cada vez que los ejecutamos. Nuestros siguientes dos algoritmos son

ejemplos de algoritmos no deterministas, donde podemos ver resultados diferentes si los ejecutamos varias veces, incluso en los mismos datos.

## Propagación de etiquetas

El algoritmo de propagación de etiquetas (LPA) es un algoritmo rápido para encontrar comunidades en un gráfico. En LPA, los nodos seleccionan su grupo en función de sus vecinos directos. Este proceso se adapta bien a las redes donde las agrupaciones son menos claras y se pueden usar ponderaciones para ayudar a un nodo a determinar en qué comunidad se ubicará. También se presta bien para el aprendizaje semisupervisado porque puede sembrar el proceso con etiquetas de nodo indicativas preasignadas.

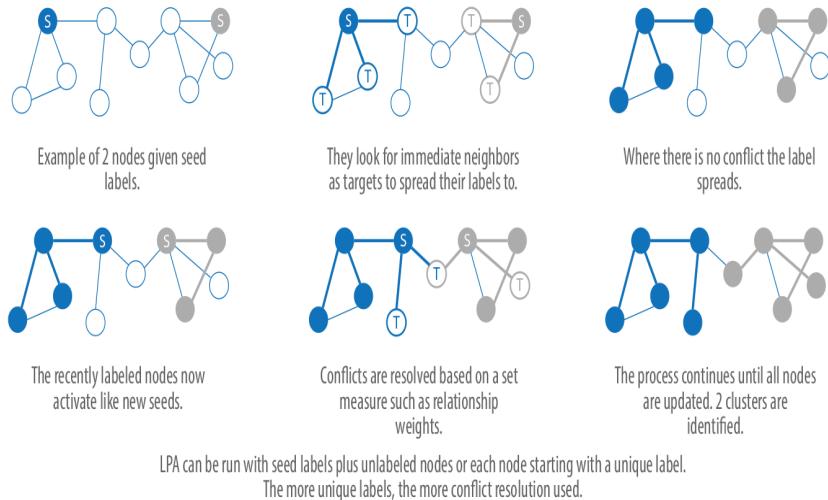
La intuición detrás de este algoritmo es que una sola etiqueta puede convertirse rápidamente en dominante en un grupo de nodos densamente conectados, pero tendrá problemas para cruzar una región escasamente conectada. Las etiquetas quedan atrapadas dentro de un grupo de nodos densamente conectados, y los nodos que terminan con la misma etiqueta cuando el algoritmo finaliza se consideran parte de la misma comunidad. El algoritmo resuelve las superposiciones, donde los nodos son potencialmente parte de varios clústeres, al asignar la pertenencia a la vecindad de la etiqueta con la relación más alta combinada y el peso del nodo.

LPA es un algoritmo relativamente nuevo propuesto en 2007 por UN Raghavan, R. Albert y S. Kumara, en un documento titulado [“Algoritmo de tiempo casi lineal para detectar estructuras de la comunidad en redes a gran escala”](#).

La Figura 6-8 muestra dos variaciones de la propagación de etiquetas, un método de inserción simple y el método de extracción más típico que se basa en los pesos de relación. El método de extracción se presta bien a la paralelización.

**Label Propagation—PUSH**

Pushes Labels to Neighbors to Find Clusters

**Label Propagation—PULL**

Pulls Labels from Neighbors Based on Relationship Weights to Find Clusters

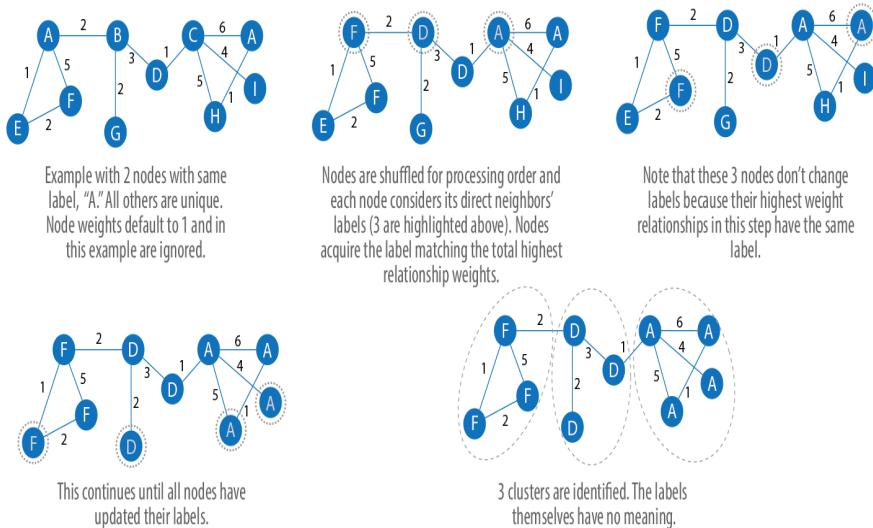


Figura 6-8. Dos variaciones de propagación de etiquetas

Los pasos que se utilizan a menudo para el método de extracción de la propagación de etiquetas son:

1. Cada nodo se inicializa con una etiqueta única (un identificador) y, opcionalmente, se pueden usar etiquetas de "semilla" preliminares.
2. Estas etiquetas se propagan a través de la red.
3. En cada iteración de propagación, cada nodo actualiza su etiqueta para que coincida con la que tiene el peso máximo, que se calcula en función de los pesos de los nodos vecinos y sus relaciones. Los lazos se rompen de manera uniforme y al azar.
4. LPA alcanza la convergencia cuando cada nodo tiene la etiqueta de la mayoría de sus vecinos.

A medida que las etiquetas se propagan, los grupos de nodos densamente conectados alcanzan rápidamente un consenso sobre una etiqueta única. Al final de la propagación, solo quedarán unas pocas etiquetas, y los nodos que tienen la misma etiqueta pertenecen a la misma comunidad.

## Aprendizaje semi-supervisado y etiquetas de semillas

A diferencia de otros algoritmos, la propagación de etiquetas puede devolver diferentes estructuras de comunidad cuando se ejecuta varias veces en el mismo gráfico. El orden en que LPA evalúa los nodos puede influir en las comunidades finales que devuelve.

El rango de soluciones se reduce cuando a algunos nodos se les dan etiquetas preliminares (es decir, etiquetas de semillas), mientras que otros no están etiquetados. Es más probable que los nodos sin etiquetar adopten las etiquetas preliminares.

Este uso de la propagación de etiquetas se puede considerar un método de aprendizaje semi-supervisado para encontrar comunidades. El aprendizaje semi-supervisado es una clase de tareas y técnicas de aprendizaje automático que operan con una pequeña cantidad de datos etiquetados, junto con una mayor cantidad de datos sin etiquetar. También podemos ejecutar el algoritmo repetidamente en los gráficos a medida que evolucionan.

Finalmente, LPA a veces no converge en una sola solución. En esta situación, los resultados de nuestra comunidad cambiarán continuamente entre algunas comunidades notablemente similares y el algoritmo nunca se completaría. Las etiquetas de semillas ayudan a guiarlo hacia una solución. Spark y Neo4j utilizan un número máximo establecido de iteraciones para evitar la ejecución interminable. Debe probar la configuración de iteración de sus datos para equilibrar la precisión y el tiempo de ejecución.

## ¿Cuándo debo usar la propagación de etiquetas?

Use la propagación de etiquetas en redes a gran escala para la detección inicial de la comunidad, especialmente cuando haya pesos disponibles. Este algoritmo puede ser paralelizado y por lo tanto es extremadamente rápido en la partición de gráficos.

Ejemplos de casos de uso incluyen:

- Asignación de polaridad de tweets como parte del análisis semántico. En este escenario, las etiquetas de semillas positivas y negativas de un clasificador se utilizan en combinación con el gráfico de seguidores de Twitter. Para obtener más información, consulte [“Clasificación de polaridad de Twitter con propagación de etiquetas sobre enlaces léxicos y el gráfico de seguidor”](#), por M. Speriosu et al.
- Encontrar combinaciones potencialmente peligrosas de posibles medicamentos co-prescritos, basados en la similitud química y los perfiles de efectos secundarios. Consulte [“Predicción de propagación de etiquetas de las interacciones entre medicamentos según los efectos secundarios clínicos”](#), un artículo de P. Zhang et al.
- Inferir las funciones de diálogo y la intención del usuario para un modelo de aprendizaje automático. Para obtener más información, consulte [“Inferencia de características basada en la propagación de etiquetas en la gráfica de Wikidata para DST”](#), un documento de Y. Murase et al.

## Propagación de etiquetas con Apache Spark

Comenzando con Apache Spark, primero importaremos los paquetes que necesitamos de Spark y el paquete de GraphFrames:

**desde pyspark.sql funciones de importación como F**

Ahora estamos listos para ejecutar el algoritmo de propagación de etiquetas. Escribimos el siguiente código para hacer esto:

```
resultado = g .labelPropagation ( maxiter = 10 ) ( resultado .especie ( "etiqueta" ) .GroupBy ( "etiqueta" ) .agg ( F .collect_list ( "id" ) ) .espectáculo ( truncar = Falso ) )
```

Si ejecutamos ese código en pyspark veremos esta salida:

etiqueta	collect_list (id)
180388626432	[jpy-core, jpy-console, jupyter]
223338299392	[matplotlib, spacy]
498216206336	[python-dateutil, numpy, six, pytz]
549755813888	[pandas]

---

 558345748480 [nbconvert, ipykernel, jpy-client]
 

---



---

 936302870528 [pyspark]
 

---



---

 1279900254208 [py4j]
 

---

En comparación con los [Componentes conectados](#), tenemos más grupos de bibliotecas en este ejemplo. LPA es menos estricto que Connected Components con respecto a cómo determina los clústeres. Se puede encontrar que dos vecinos (nodos conectados directamente) están en diferentes agrupaciones utilizando la propagación de etiquetas. Sin embargo, al usar Componentes Conectados, un nodo siempre estará en el mismo grupo que sus vecinos porque ese algoritmo basa el agrupamiento estrictamente en las relaciones.

En nuestro ejemplo, la diferencia más obvia es que las bibliotecas de Jupyter se han dividido en dos comunidades: una que contiene las partes centrales de la biblioteca y la otra las herramientas orientadas al cliente.

## Propagación de etiquetas con Neo4j

Ahora probemos el mismo algoritmo con Neo4j. Podemos ejecutar LPA ejecutando la siguiente consulta:

```
CALL algo.labelPropagation.stream ( "Library" , "DEPENDS_ON" , { iterations : 10}) IIDEL nodeId, label RETURN label, collect (algo.getNodeById (nodeId). Id ) AS bibliotecas ORDER BY tamaño (librerías) DESC
```

Los parámetros pasados a este algoritmo son:

Biblioteca

La etiqueta de nodo para cargar desde el gráfico.

DEPENDE DE

El tipo de relación a cargar desde el gráfico.

iteraciones: 10

El número máximo de iteraciones para ejecutar

Estos son los resultados que veríamos:

---

**etiqueta bibliotecas**

---



---

 11 [matplotlib, spacy, six, pandas, python-dateutil]
 

---



---

 10 [jupyter, jpy-console, nbconvert, jpy-client, jpy-core]
 

---



---

 4 [pyspark, py4j]
 

---



---

 8 [ipykernel]
 

---



---

 13 [numpy]
 

---



---

 0 [pytz]
 

---

Los resultados, que también se pueden ver visualmente en la [Figura 6-9](#), son bastante similares a los que obtuvimos con Apache Spark.

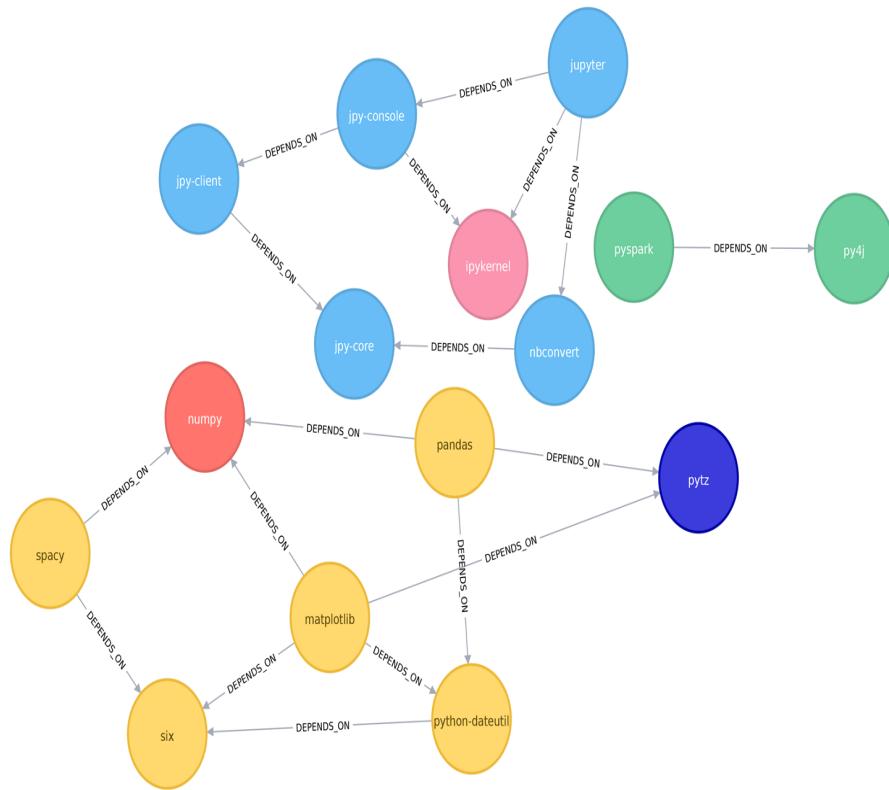


Figura 6-9. Clusters encontrados por el algoritmo de propagación de etiquetas

También podemos ejecutar el algoritmo asumiendo que el gráfico no está dirigido, lo que significa que los nodos intentarán adoptar etiquetas de las bibliotecas de las que dependen, así como las que dependen de ellas. Para hacer esto, pasamos el parámetro DIRECCIÓN: AMBOS al algoritmo:

```
LLAMADA algo.labelPropagation.stream ( "biblioteca" , "DEPENDS_ON" , {iteraciones: 10, dirección: "ambos" }) NODEID
RENDIMIENTO, etiqueta RETURN etiqueta, recoger (. Algo.getNodeById (NodeID) Identificación ) AS bibliotecas ORDER BY
tamaño (bibliotecas) DESC
```

Si ejecutamos eso, obtendremos la siguiente salida:

#### **etiqueta bibliotecas**

11	[Pytz, matplotlib, spacy, six, pandas, numpy, python-dateutil]
10	[nbconvert, jpy-client, jpy-core]
6	[jupyter, jpy-console, ipykernel]
4	[pyspark, py4j]

El número de grupos se ha reducido de seis a cuatro, y todos los nodos en la parte matplotlib del gráfico ahora están agrupados. Esto se puede ver más claramente en la [Figura 6-10](#).

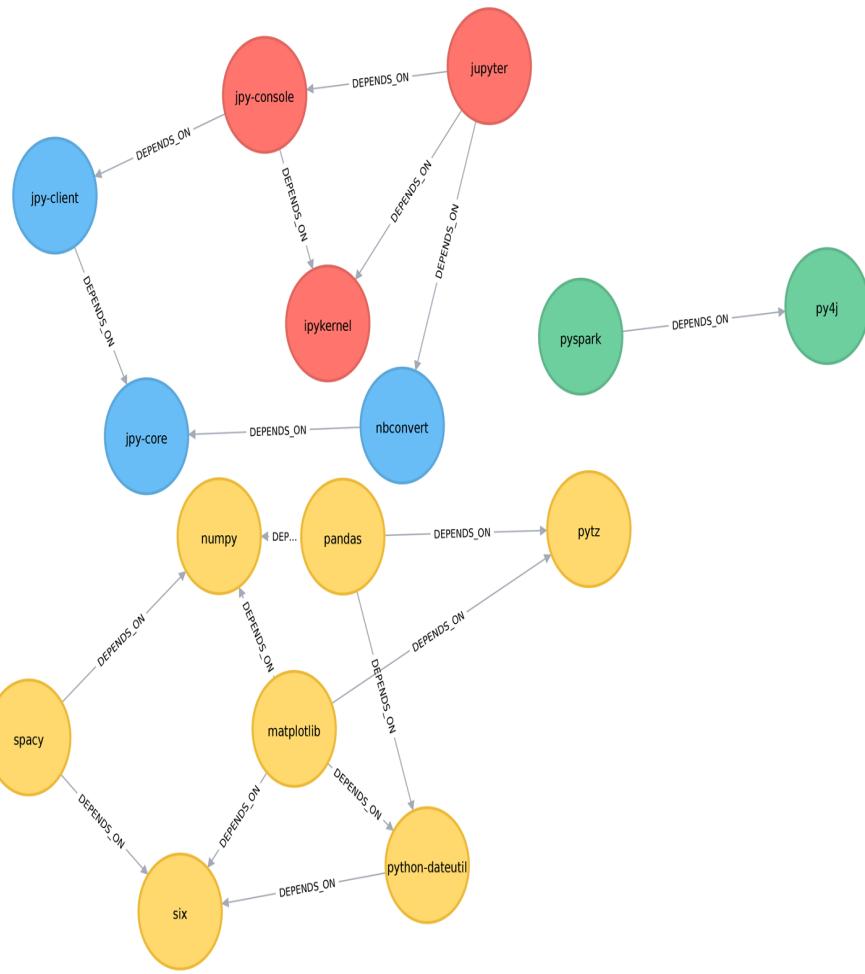


Figura 6-10. Grupos encontrados por el algoritmo de propagación de etiquetas al ignorar la dirección de la relación

Aunque los resultados de la ejecución de la propagación de etiquetas en estos datos son similares para los cálculos dirigidos y no dirigidos, en gráficos complicados verás diferencias más significativas. Esto se debe a que ignorar la dirección hace que los nodos intenten adoptar más etiquetas, independientemente de la fuente de la relación.

## Modularidad de Lovaina

El algoritmo de modularidad de Louvain encuentra grupos comparando la densidad de la comunidad al asignar nodos a diferentes grupos. Puede considerar esto como un análisis "qué pasaría si" para probar varias agrupaciones con el objetivo de alcanzar un óptimo global.

Propuesto en 2008, el [algoritmo de Louvain](#) es uno de los algoritmos basados en modularidad más rápidos. Además de detectar comunidades, también revela una jerarquía de comunidades en diferentes escalas. Esto es útil para comprender la estructura de una red en diferentes niveles de granularidad.

Louvain cuantifica qué tan bien se asigna un nodo a un grupo al observar la densidad de las conexiones dentro de un grupo en comparación con una muestra promedio o aleatoria. Esta medida de la asignación de la comunidad se llama modularidad .

### Agrupación basada en la calidad vía modularidad.

La modularidad es una técnica para descubrir comunidades al dividir una gráfica en módulos (o agrupaciones) de grano más grueso y luego medir la fuerza de las agrupaciones. A diferencia de solo observar la concentración de conexiones dentro de un grupo, este método compara las densidades de relaciones en grupos dados con las densidades entre grupos. La medida de la calidad de esas agrupaciones se denomina modularidad.

Los algoritmos de modularidad optimizan las comunidades a nivel local y luego a nivel mundial, utilizando múltiples iteraciones para probar diferentes agrupaciones y aumentar la tosiedad. Esta estrategia identifica

las jerarquías de la comunidad y proporciona una amplia comprensión de la estructura general. Sin embargo, todos los algoritmos de modularidad adolecen de dos inconvenientes:

- Fusionan comunidades más pequeñas en otras más grandes.
- Se puede producir una meseta donde existen varias opciones de partición con una modularidad similar, formando máximos locales e impidiendo el progreso.

Para obtener más información, consulte el artículo "[El rendimiento de la maximización de la modularidad en contextos prácticos](#)" , por BH Good, Y.-A. de Montjoye, y A. Clauset.

## CÁLCULO DE LA MODULARIDAD

Un cálculo simple de modularidad se basa en la fracción de las relaciones dentro de los grupos dados menos la fracción esperada si las relaciones se distribuyen al azar entre todos los nodos. El valor siempre está entre 1 y -1, con valores positivos que indican una mayor densidad de relación de lo que cabría esperar por casualidad y valores negativos que indican una menor densidad. [La Figura 6-11](#) ilustra varias puntuaciones de modularidad diferentes basadas en agrupaciones de nodos.

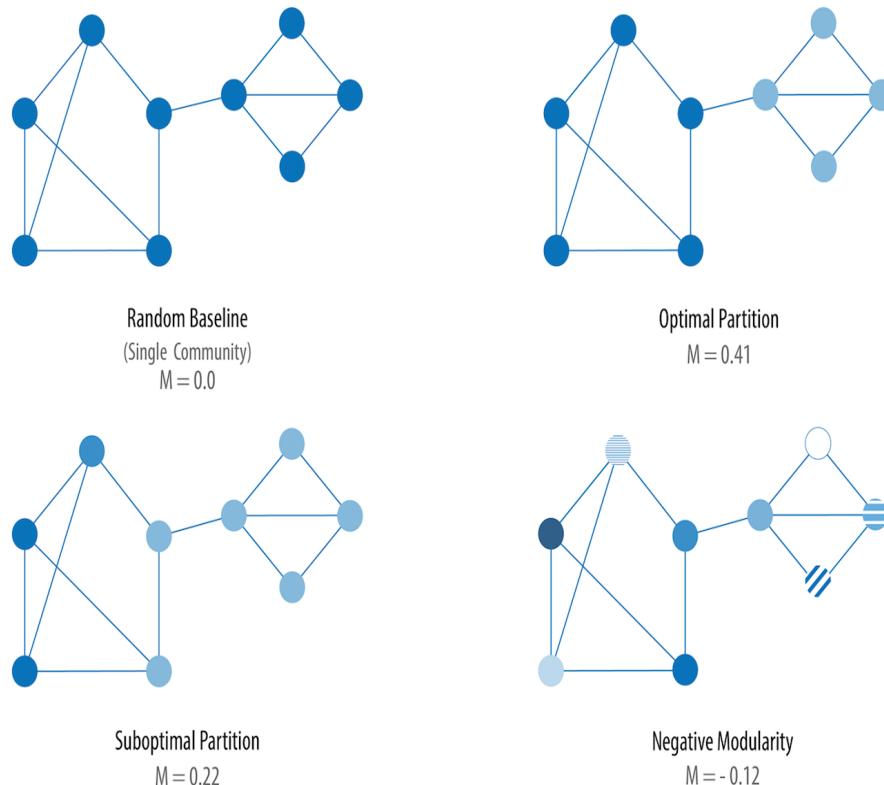


Figura 6-11. Cuatro puntuaciones de modularidad basadas en diferentes opciones de partición

La fórmula para la modularidad de un grupo es:

$$M = \sum_{c=1}^n \frac{L_c - k_c}{2L}$$

dónde:

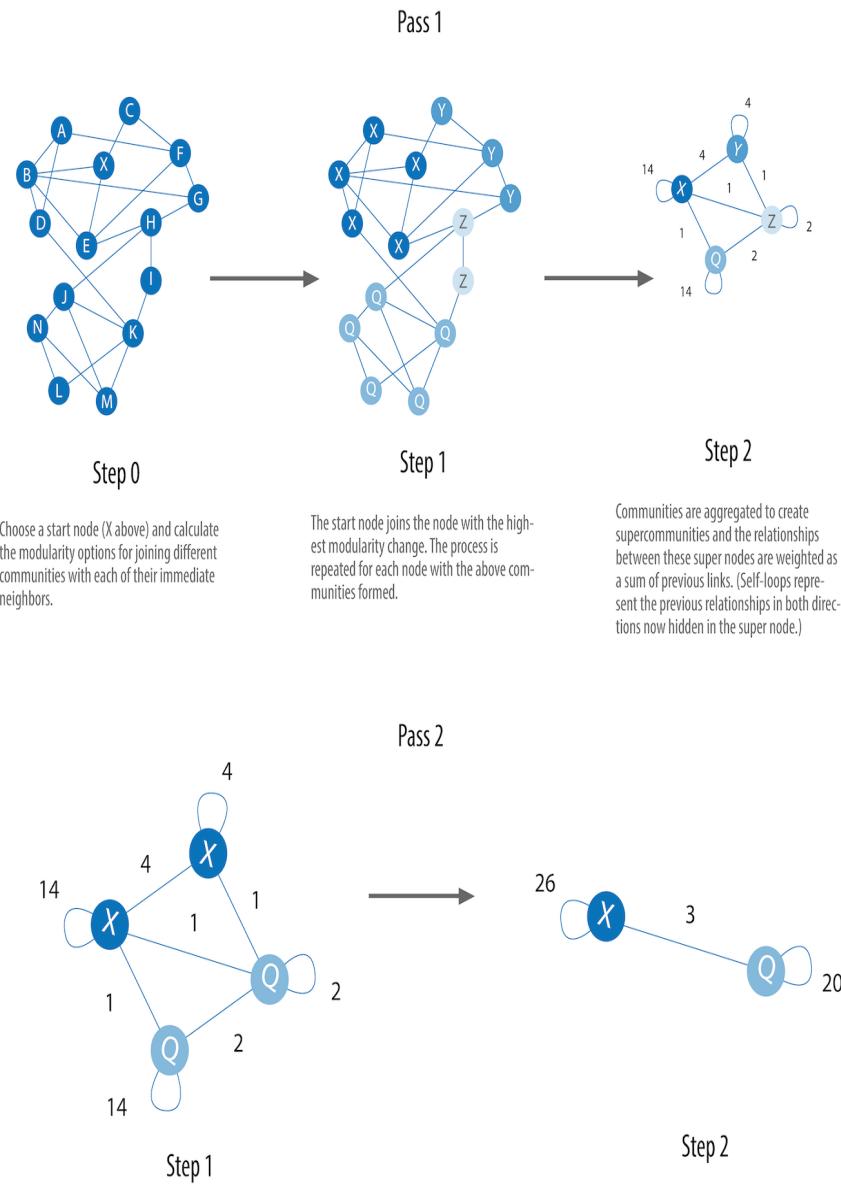
- L es el número de relaciones en todo el grupo.
- L<sub>c</sub> es el número de relaciones en una partición.
- K<sub>c</sub> es el grado total de nodos en una partición.

El cálculo para la partición óptima en la parte superior de la [Figura 6-11](#) es el siguiente:

- La partición oscura es  $\frac{7}{13} - \frac{15}{2(13)} = 0.205$
- La partición de luz es  $\frac{5}{13} - \frac{11}{2(13)} = 0.206$
- Estos se suman para  $M = 0.205 + 0.206 = 0.41$

Inicialmente, el algoritmo de modularidad de Louvain optimiza la modularidad localmente en todos los nodos, lo que encuentra pequeñas comunidades; luego, cada pequeña comunidad se agrupa en un nodo conglomerado más grande y el primer paso se repite hasta alcanzar un óptimo global.

El algoritmo consiste en la aplicación repetida de dos pasos, como se ilustra en la [Figura 6-12](#).



Steps 1 and 2 repeat in passes until there is no further increase in modularity or a set number of iterations have occurred.

Figura 6-12. El proceso del algoritmo de Lovaina.

Los pasos del algoritmo de Lovaina incluyen:

1. Una asignación “codicosa” de nodos a comunidades, favoreciendo optimizaciones locales de modularidad.
2. La definición de una red más amplia basada en las comunidades que se encuentran en el primer paso. Esta red de grano grueso se utilizará en la siguiente iteración del algoritmo.

Estos dos pasos se repiten hasta que no sean posibles más reasignaciones de comunidades que aumenten la modularidad.

Parte del primer paso de optimización es evaluar la modularidad de un grupo. Louvain usa la siguiente fórmula para lograr esto:

$$Q = \frac{1}{2m} \sum_{u, v} A_{uv} - \frac{k_u k_v}{2m} \delta(c_u, c_v)$$

dónde:

- U y v son nodos.

- $M$  es el peso total de la relación en todo el gráfico (  $2m$  es un valor de normalización común en las fórmulas de modularidad).
- $A_{uv} - \frac{k_u k_v}{2m}$  es la fuerza de la relación entre  $u$  y  $v$  en comparación con lo que cabría esperar con una asignación aleatoria (tiende a promedios) de esos nodos en la red.
  - $A_{uv}$  es el peso de la relación entre  $u$  y  $v$  .
  - $k_u$  es la suma de los pesos de relación para  $u$  .
  - $k_v$  es la suma de los pesos de relación para  $v$  .
- $\Delta(c_u, c_v)$  es igual a 1 si  $u$  y  $v$  se asignan a la misma comunidad, y 0 si no lo son.

Otra parte de ese primer paso evalúa el cambio en la modularidad si un nodo se mueve a otro grupo. Louvain usa una variación más complicada de esta fórmula y luego determina la mejor asignación de grupo.

## ¿Cuándo debo usar Louvain?

Utilice la modularidad de Louvain para encontrar comunidades en redes extensas. Este algoritmo aplica una heurística, en oposición a la modularidad exacta, que es computacionalmente costosa. Por lo tanto, Louvain se puede usar en gráficos grandes donde los algoritmos de modularidad estándar pueden tener problemas.

Louvain también es muy útil para evaluar la estructura de redes complejas, en particular, para descubrir muchos niveles de jerarquías, como lo que podría encontrar en una organización criminal. El algoritmo puede proporcionar resultados en los que puede acercarse a diferentes niveles de granularidad y encontrar subcomunidades dentro de subcomunidades dentro de subcomunidades.

Ejemplos de casos de uso incluyen:

- Detección de ciberataques. El algoritmo de Lovaina se utilizó en [un estudio de 2016 realizado por SV Shanbhag](#) sobre detección rápida de comunidades en [redes](#) ciberneticas a gran escala para aplicaciones de ciberseguridad. Una vez que se han detectado estas comunidades, se pueden utilizar para detectar ataques ciberneticos.
- Extraer temas de plataformas sociales en línea, como Twitter y YouTube, en función de la co-ocurrencia de términos en documentos como parte del proceso de modelado de temas. Este enfoque se describe en un artículo de GS Kido, RA Igawa y S. Barbon Jr., ["Modelado de temas basados en el método de Louvain en las redes sociales en línea"](#) .
- Encontrar estructuras de comunidad jerárquicas dentro de la red funcional del cerebro, como se describe en ["Modularidad jerárquica en redes funcionales del cerebro humano"](#) por D. Meunier et al.

### ADVERTENCIA

Los algoritmos de optimización de modularidad, incluyendo Louvain, tienen dos problemas. Primero, los algoritmos pueden pasar por alto pequeñas comunidades dentro de grandes redes. Puede superar este problema revisando los pasos de consolidación intermedios. Segundo, en los gráficos grandes con comunidades superpuestas, los optimizadores de modularidad pueden no determinar correctamente los máximos globales. En este último caso, recomendamos utilizar cualquier algoritmo de modularidad como una guía para la estimación bruta pero no la precisión completa.

## Lovaina con Neo4j

Veamos el algoritmo de Lovaina en acción. Podemos ejecutar la siguiente consulta para ejecutar el algoritmo sobre nuestro gráfico:

```
CALL algo.louvain.stream( "Library" , "DEPENDS_ON" ) YIELD nodeId, comunidades RETURN algo.getNodeById(nodeId).
```

Identificación del AS bibliotecas, comunidades

Los parámetros pasados a este algoritmo son:

Biblioteca

La etiqueta de nodo para cargar desde el gráfico.

DEPENDE DE

El tipo de relación a cargar desde el gráfico.

Estos son los resultados:

<b>bibliotecas</b>	<b>comunidades</b>
pytz	[0, 0]
pyspark	[1, 1]
matplotlib	[2, 0]
espacioso	[2, 0]
py4j	[1, 1]
jupyter	[3, 2]
jpy-console	[3, 2]
nbconvert	[4, 2]
ipykernel	[3, 2]
cliente jpy	[4, 2]
jpy-core	[4, 2]
seis	[2, 0]
pandas	[0, 0]
adormecido	[2, 0]
python-dateutil	[2, 0]

La columna de comunidades describe la comunidad en la que se encuentran los nodos en dos niveles. El último valor en la matriz es la comunidad final y la otra es una comunidad intermedia.

Los números asignados a las comunidades intermedias y finales son simplemente etiquetas sin un significado medible. Considérelos como etiquetas que indican a qué nodos de comunidad pertenecen, por ejemplo, "pertenece a una comunidad etiquetada con 0", "una comunidad con etiqueta 4", etc.

Por ejemplo, matplotlib tiene un resultado de [2,0] . Esto significa que la comunidad final de matplotlib está etiquetada como 0 y su comunidad intermedia está etiquetada como 2 .

Es más fácil ver cómo funciona esto si almacenamos estas comunidades utilizando la versión de escritura del algoritmo y luego las consultamos. La siguiente consulta ejecutará el algoritmo de Louvain y almacenará el resultado en la propiedad de las comunidades en cada nodo:

```
CALL algo.louvain ( "Library" , "DEPENDS_ON" )
```

También podríamos almacenar las comunidades resultantes utilizando la versión de transmisión del algoritmo, y luego llamar a la cláusula SET para almacenar el resultado. La siguiente consulta muestra cómo podríamos hacer esto:

```
CALL algo.louvain.stream ( "Library" , "DEPENDS_ON" ) YIELD nodeId, comunidades WITH algo.getNodeById (nodeId) AS node , comunidades SET node .communities = comunidades
```

Una vez que hayamos ejecutado cualquiera de esas consultas, podemos escribir la siguiente consulta para encontrar los clústeres finales:

```
PARTIDO (l: Biblioteca) DEVOLVER l.communities [-1] AS community, recopilar (l. Id ) AS bibliotecas ORDENAR POR tamaño (bibliotecas) DESC
```

l.communities [-1] devuelve el último elemento de la matriz subyacente que almacena esta propiedad.

Al ejecutar la consulta se obtiene este resultado:

<b>comunidad</b>	<b>bibliotecas</b>
0	[Pytz, matplotlib, spacy, six, pandas, numpy, python-dateutil]

---

2 [jupyter, jpy-console, nbconvert, ipykernel, jpy-client, jpy-core]

---

1 [pyspark, py4j]

Este agrupamiento es el mismo que vimos con el algoritmo de componentes conectados.

matplotlib está en una comunidad con pytz, spacy, six, pandas, numpy y python-dateutil. Podemos ver esto más claramente en la [Figura 6-13](#).

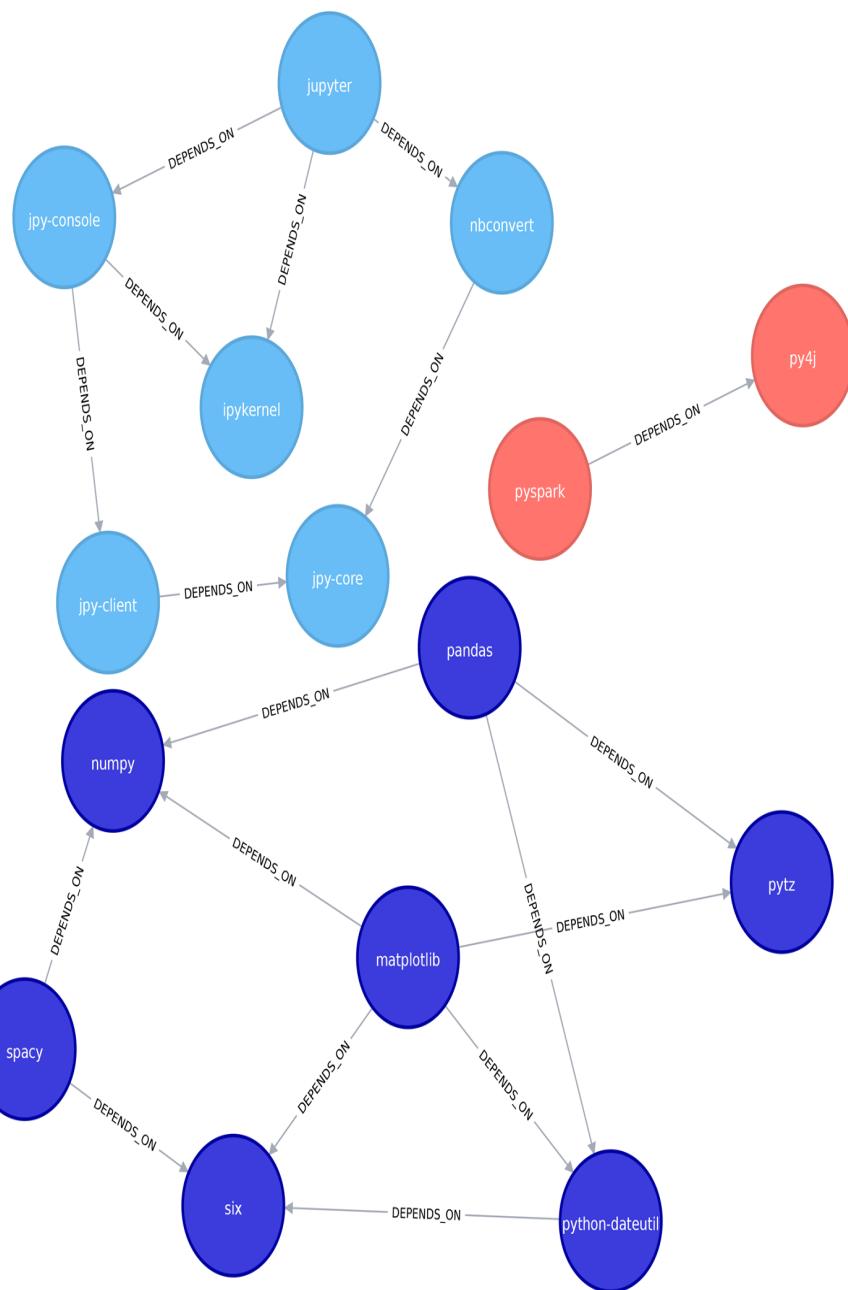


Figura 6-13. Grupos encontrados por el algoritmo de Louvain

Una característica adicional del algoritmo de Lovaina es que también podemos ver el agrupamiento intermedio. Esto nos mostrará grupos más finos que la capa final:

MATCH (l: Library) RETURN l.communities [0] AS community, recopile (l. Id ) AS bibliotecas ORDENADO POR tamaño (bibliotecas) DESC

Ejecutando esa consulta da esta salida:

---

**comunidad bibliotecas**

---

2 [matplotlib, spacy, seis, python-dateutil]

---

4 [nbconvert, jpy-client, jpy-core]

---

3 [jupyter, jpy-console, ipykernel]

1	[pyspark, py4j]
0	[Pytz, pandas]
5	[numpy]

Las bibliotecas en la comunidad matplotlib ahora se han dividido en tres comunidades más pequeñas:

- Matplotlib, spacy, six y python-dateutil
- Pytz y pandas
- Entumecido

Podemos ver este desglose visualmente en la [Figura 6-14](#).

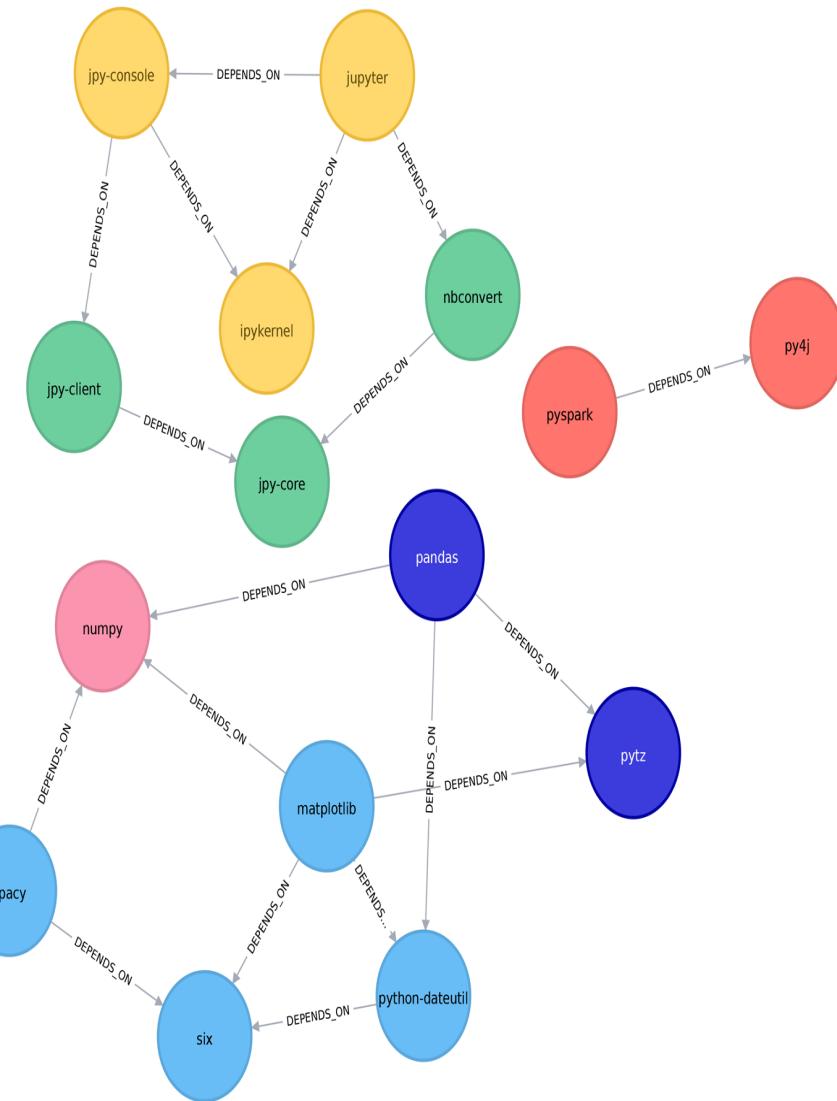


Figura 6-14. Grupos intermedios encontrados por el algoritmo de Lovaina

Aunque este gráfico solo mostró dos capas de jerarquía, si ejecutáramos este algoritmo en un gráfico más grande, veríamos una jerarquía más compleja. Los grupos intermedios que Louvain revela pueden ser muy útiles para detectar comunidades de grano fino que pueden no ser detectadas por otros algoritmos de detección de la comunidad..

## Validando Comunidades

Los algoritmos de detección de la comunidad generalmente tienen el mismo objetivo: identificar grupos. Sin embargo, debido a que diferentes algoritmos comienzan con diferentes suposiciones, pueden descubrir diferentes comunidades. Esto hace que elegir el algoritmo correcto para un problema en particular sea más desafiante y un poco de exploración.

La mayoría de los algoritmos de detección comunitaria funcionan razonablemente bien cuando la densidad de las relaciones es alta dentro de los grupos en comparación con su entorno, pero las redes del mundo real a menudo son menos distintas. Podemos validar la precisión de las comunidades encontradas comparando nuestros resultados con un punto de referencia basado en datos con comunidades conocidas.

Dos de los puntos de referencia más conocidos son los algoritmos Girvan-Newman (GN) y Lancichinetti – Fortunato – Radicchi (LFR). Las redes de referencia que generan estos algoritmos son bastante diferentes: GN genera una red aleatoria que es más homogénea, mientras que LFR crea un gráfico más heterogéneo donde los grados de nodo y el tamaño de la comunidad se distribuyen de acuerdo con una ley de potencia.

Dado que la precisión de nuestras pruebas depende del punto de referencia utilizado, es importante que nuestro punto de referencia coincida con nuestro conjunto de datos. En la medida de lo posible, busque densidades similares, distribuciones de relaciones, definiciones de comunidad y dominios relacionados.

## Resumen

Los algoritmos de detección de la comunidad son útiles para comprender la forma en que los nodos se agrupan en una gráfica.

En este capítulo, comenzamos por aprender acerca de los algoritmos de Conteo de triángulos y Coeficiente de agrupamiento. Luego pasamos a dos algoritmos de detección de la comunidad determinista: Componentes fuertemente conectados y Componentes conectados. Estos algoritmos tienen definiciones estrictas de lo que constituye una comunidad y son muy útiles para familiarizarse con la estructura del gráfico al principio de la línea de análisis de gráficos.

Terminamos con Label Propagation y Louvain, dos algoritmos no deterministas que son más capaces de detectar comunidades de grano más fino. Lovaina también nos mostró una jerarquía de comunidades en diferentes escalas.

En el siguiente capítulo, tomaremos un conjunto de datos mucho más grande y aprenderemos cómo combinar los algoritmos para obtener una mejor visión de nuestros datos conectados.

# Capítulo 7. Gráficos de algoritmos en la práctica.

---

El enfoque que tomamos para graficar el análisis evoluciona a medida que nos familiarizamos con el comportamiento de diferentes algoritmos en conjuntos de datos específicos. En este capítulo, veremos varios ejemplos para ofrecerle una mejor idea de cómo abordar el análisis de datos de gráficos a gran escala utilizando conjuntos de datos de Yelp y el Departamento de Transporte de EE. UU. Veremos el análisis de datos de Yelp en Neo4j que incluye una descripción general de los datos, combinando algoritmos para hacer recomendaciones de viaje, y extrayendo datos de usuarios y de negocios para consultar. En Spark, analizaremos los datos de las aerolíneas de EE. UU. Para comprender los patrones de tráfico y los retrasos, así como la forma en que los aeropuertos están conectados por diferentes aerolíneas.

Debido a que los algoritmos de búsqueda de rutas son sencillos, nuestros ejemplos utilizarán estos algoritmos de detección de centralidad y comunidad:

- PageRank para encontrar revisores de Yelp influyentes y luego correlacionar sus calificaciones para hoteles específicos
- Centralidad de intermediación para descubrir revisores conectados a múltiples grupos y luego extraer sus preferencias.
- Propagación de etiquetas con una proyección para crear supercategorías de empresas similares de Yelp
- Degree Centrality para identificar rápidamente los centros de aeropuertos en el conjunto de datos de transporte de EE. UU.
- Componentes fuertemente conectados para observar grupos de rutas de aeropuertos en los EE. UU.

## Analizando datos de Yelp con Neo4j

Yelp ayuda a las personas a encontrar negocios locales según las reseñas, preferencias y recomendaciones. A finales de 2018, se habían escrito más de 180 millones de revisiones en la plataforma. Desde 2013, Yelp ha ejecutado el [desafío del conjunto de datos de Yelp](#), una competencia que alienta a las personas a explorar e investigar el conjunto de datos abierto de Yelp.

A partir de la Ronda 12 (realizada en 2018) del desafío, el conjunto de datos abierto contenía:

- Más de 7 millones de comentarios más consejos.
- Más de 1.5 millones de usuarios y 280,000 fotos.
- Más de 188,000 empresas con 1.4 millones de atributos.
- 10 áreas metropolitanas

Desde su lanzamiento, el conjunto de datos se ha vuelto popular, con [cientos de documentos académicos](#) escritos utilizando este material. El conjunto de datos de Yelp representa datos reales que están muy bien estructurados y altamente interconectados. Es un gran escaparate de algoritmos de gráficos que también puede descargar y explorar.

## Red social de Yelp

Además de escribir y leer comentarios sobre empresas, los usuarios de Yelp forman una red social. Los usuarios pueden enviar solicitudes de amistad a otros usuarios que han encontrado mientras navegan por Yelp.com, o pueden conectar sus libretas de direcciones o gráficos de Facebook.

El conjunto de datos de Yelp también incluye una red social. [La Figura 7-1](#) es una captura de pantalla de la sección Amigos del perfil de Yelp de Mark.

Figura 7-1. El perfil de Mark en Yelp

A parte del hecho de que Mark necesita algunos amigos más, estamos listos para comenzar. Para ilustrar cómo podemos analizar los datos de Yelp en Neo4j, usaremos un escenario donde trabajamos para un negocio de información de viajes. Primero exploraremos los datos de Yelp y luego veremos cómo ayudar a las personas a planificar viajes con nuestra aplicación. Caminaremos para encontrar buenas recomendaciones de lugares para hospedarse y cosas que hacer en las principales ciudades como Las Vegas.

Otra parte de nuestro escenario de negocios involucrará la consultoría para negocios de destino de viaje. En un ejemplo, ayudaremos a los hoteles a identificar visitantes influyentes y luego a empresas a las que deberían dirigirse para programas de promoción cruzada.

## Importación de datos

Existen muchos métodos diferentes para importar datos a Neo4j, incluida la [herramienta de importación](#), el [comando LOAD CSV](#) que hemos visto en capítulos anteriores y los [controladores Neo4j](#).

Para el conjunto de datos de Yelp debemos realizar una importación única de una gran cantidad de datos, por lo que la herramienta de importación es la mejor opción. Consulte "[Neo4j Bulk Data Import and Yelp](#)" para obtener más detalles.

## Modelo gráfico

Los datos de Yelp se representan en un modelo gráfico como se muestra en la [Figura 7-2](#).

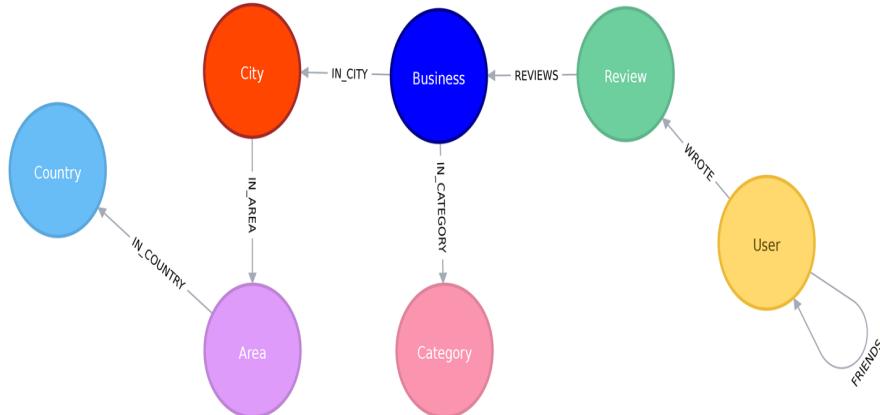


Figura 7-2. El modelo gráfico de Yelp

Nuestro gráfico contiene nodos etiquetados por el usuario, que tienen relaciones de AMIGOS con otros usuarios. Los usuarios también escriben reseñas y consejos sobre negocios. Todos los metadatos se almacenan como propiedades de nodos, a excepción de las categorías comerciales, que están representadas por nodos de Categoría separados. Para los datos de ubicación, hemos extraído los atributos de Ciudad, Área

y País en el subgrafo. En otros casos de uso, podría tener sentido extraer otros atributos a nodos como fechas o colapsar nodos a relaciones como revisiones.

El conjunto de datos de Yelp también incluye sugerencias y fotos para el usuario, pero no las usaremos en nuestro ejemplo.

## Una descripción rápida de los datos de Yelp

Una vez que tengamos los datos cargados en Neo4j, ejecutaremos algunas consultas exploratorias.

Preguntaremos cuántos nodos hay en cada categoría o qué tipos de relaciones existen, para tener una idea de los datos de Yelp. Anteriormente, mostramos las consultas de Cypher para nuestros ejemplos de Neo4j, pero podríamos estar ejecutándolos desde otro lenguaje de programación. Como Python es el lenguaje de referencia para los científicos de datos, usaremos el controlador Python de Neo4j en esta sección cuando deseamos conectar los resultados a otras bibliotecas del ecosistema de Python. Si solo queremos mostrar el resultado de una consulta, usaremos Cypher directamente.

También mostraremos cómo combinar Neo4j con la popular biblioteca de pandas, que es efectiva para el manejo de datos fuera de la base de datos. Veremos cómo usar la biblioteca tabulada para pretender los resultados que obtenemos de los pandas y cómo crear representaciones visuales de datos utilizando matplotlib.

También utilizaremos la biblioteca de procedimientos APOC de Neo4j para ayudarnos a escribir consultas Cypher aún más potentes. Hay más información sobre APOC en ["APOC y otras herramientas Neo4j"](#).

Primero instalaremos las bibliotecas de Python:

pip instalar neo4j-driver tabular pandas matplotlib

Una vez que hayamos hecho eso, importaremos esas bibliotecas:

**de neo4j.v1 importación de base de datos orientada a grafos de importación pandas como pd de tabular importación tabular**

Importar matplotlib puede ser complicado en macOS, pero las siguientes líneas deberían hacer el truco:

```
importar matplotlib matplotlib . usar ( 'TkAgg' ) importar matplotlib.pyplot como plt
```

Si está ejecutando en otro sistema operativo, la línea media puede no ser necesaria. Ahora vamos a crear una instancia del controlador Neo4j que apunta a una base de datos Neo4j local:

```
driver = GraphDatabase . controlador ( "bolt: // localhost" , auth = ( "neo4j" , "neo" ))
```

### NOTA

Deberá actualizar la inicialización del controlador para usar su propio host y sus credenciales.

Para comenzar, veamos algunos números generales para nodos y relaciones. El siguiente código calcula las cardinalidades de las etiquetas de nodo (es decir, cuenta el número de nodos para cada etiqueta) en la base de datos:

```
result = { "label" : [] , "count" : [] } with driver . session () as session : labels = [ row [ "label" ] para row in session . ejecutar ( "CALL db.labels ()" ) para la etiqueta en las etiquetas : query = f "MATCH (:{label}) RETURN count (*) as count" count = session . ejecutar ( consulta ) . single () [ "count" ] resultado [ "label" ] . añadir ( etiqueta ) el resultado [ "contar" ] . añadir ( contar ) df = pd . DataFrame ( datos = resultado ) print ( tabular ( df . Sort_values ( "count" ), headers = 'keys' , tablefmt = 'psql' , showindex = Falso ))
```

Si ejecutamos ese código, veremos cuántos nodos tenemos para cada etiqueta:

etiqueta	contar
País	17
Zona	54
Ciudad	1093
Categoría	1293

Negocio	174567
Usuario	1326101
revisión	5261669

También podríamos crear una representación visual de las cardinalidades, con el siguiente código:

```
PLT . estilo . use ( 'fivethirtyeight' ) ax = df . plot ( kind = 'bar' , x = 'label' , y = 'count' , legend = None ) ax . xaxis . set_label_text ( "" ) plt . yscale ( "log" ) plt . xticks ( rotación = 45 ) plt . tight_layout () plt . mostrar ()
```

Podemos ver la tabla que se genera con este código en la [Figura 7-3](#). Tenga en cuenta que este gráfico está utilizando la escala de registro.

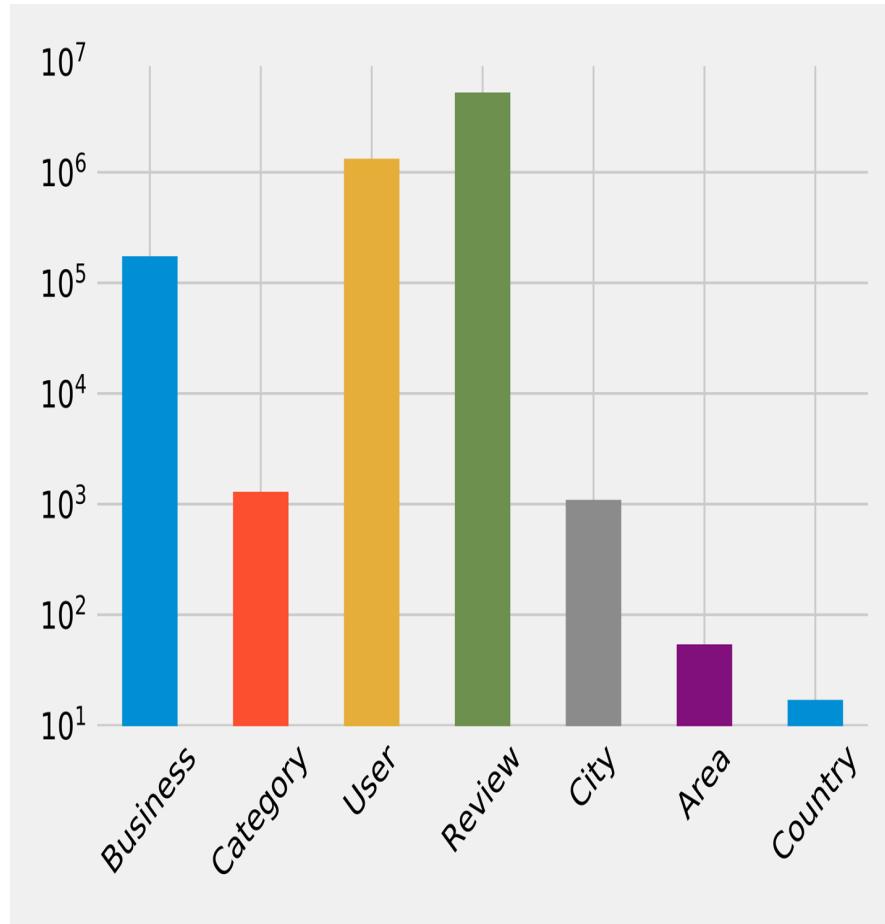


Figura 7-3. El número de nodos para cada categoría de etiqueta

Del mismo modo, podemos calcular las cardinalidades de las relaciones:

```
resultado = { "relType" : [] , "count" : [] } con el controlador . session () as session : rel_types = [ row [ "relationshipType" ] para row en sesión . run ( "CALL db.relationshipTypes ()" )] para rel_type en rel_types : query = f "MATCH () - [ : {rel_type} ` ] -> () RETURN count (*) como count" count = session . plazo ( consulta ) . single () [ "count" ] resultado [ "relType" ] . añadir ( rel_type ) resultado [ "cuenta" ] . añadir ( contar ) df = pd . DataFrame ( datos = resultado ) print ( tabular ( df . Sort_values ( "count" ), headers = 'keys' , tablefmt = 'psql' , showindex = False ))
```

Si ejecutamos ese código veremos el número de cada tipo de relación:

relType	contar
EN EL PAIS	54
EN LA ZONA	1154
EN CIUDAD	174566
IN_CATEGORY	667527
ESCRIBIR	5261669
REVISIONES	5261669

Podemos ver un gráfico de las cardinalidades en la [Figura 7-4](#). Al igual que con el gráfico de cardinalidades de nodo, este gráfico utiliza una escala de registro.

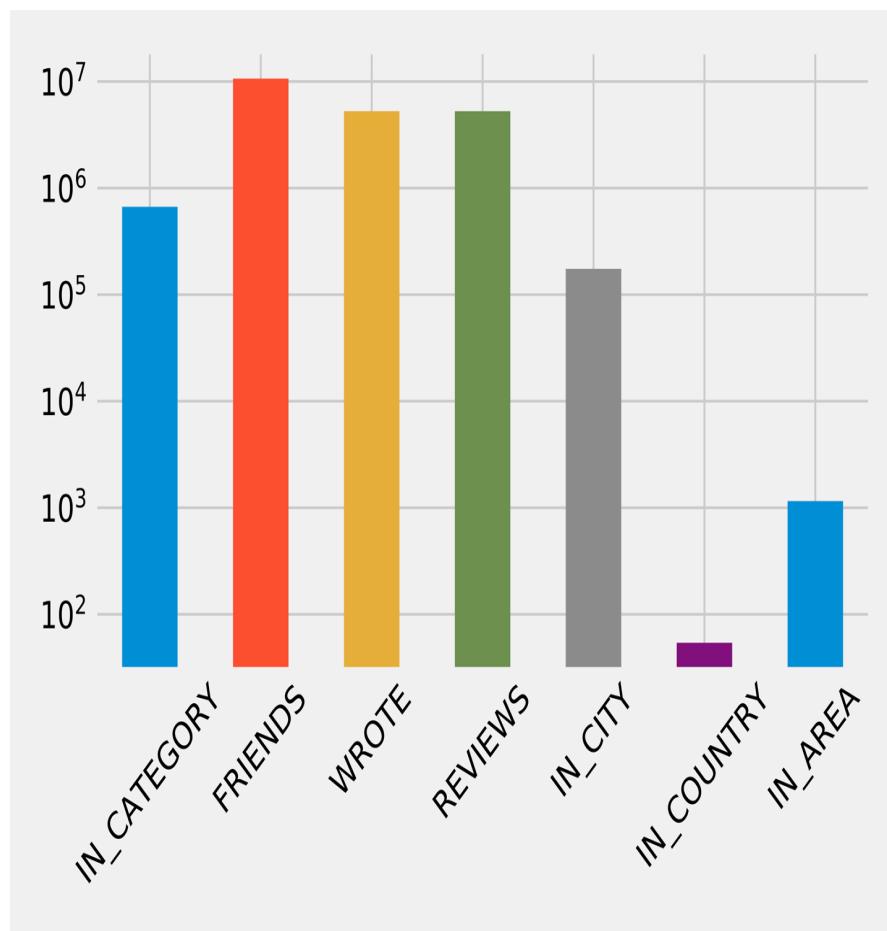


Figura 7-4. El número de relaciones por tipo de relación

Estas consultas no deben revelar nada sorprendente, pero son útiles para tener una idea de lo que hay en los datos. Esto también sirve como una comprobación rápida de que los datos importados correctamente.

Suponemos que Yelp tiene muchas reseñas de hoteles, pero tiene sentido verificar antes de que nos enfoquemos en ese sector. Podemos averiguar cuántas empresas de hoteles hay en esos datos y cuántas revisiones tienen ejecutando la siguiente consulta:

```
PARTIDO (categoría: Categoría {nombre: "Hoteles" }) RETURN size ((category) <- [: IN_CATEGORY] - () AS business, size ((: Review) - [: REVIEWS] -> (: Business) - [: IN_CATEGORY] -> (categoría) AS comentarios)
```

Aquí está el resultado:

#### negocios opiniones

2683	183759
------	--------

Tenemos muchos negocios con los que trabajar, ¡y muchas revisiones! En la siguiente sección exploraremos más los datos con nuestro escenario empresarial.

## Aplicación de planificación de viaje

Para agregar recomendaciones populares a nuestra aplicación, comenzamos por encontrar los hoteles mejor calificados como una heurística para las opciones populares para las reservas. Podemos agregar qué tan bien han sido calificados para entender la experiencia real. Para ver los 10 hoteles más revisados y trazar sus distribuciones de calificación, usamos el siguiente código:

```
# Encuentre los 10 hoteles con la mayor cantidad de comentarios query = "" " MATCH (revisión: Revisión) - [: REVIEWS] -> (business: Business), (business) - [: IN_CATEGORY] -> (category: Category {name: $ category}), (business) - [: IN_CITY] -> (: City {name: $ city}) RETURN business.name AS business, collect (review.stars) AS allReviews ORDER BY size (allReviews) DESC
```

```

LIMIT 10 """ fig = plt . figura () fig . set_size_inches ( 10.5 , 14.5 ) fig . subplots_adjust ( hspace = 0.4 , wspace = 0.4 ) con
conductor . session () as session : params = { "city" : "Las Vegas" , "category" : "Hotels" } result = session . ejecutar ( consulta ,
parámetros ) para índice , fila en enumerar ( resultado ): negocio = fila [ "negocio" ] estrellas = pd .
"allReviews" ]) total = estrellas . count () average_stars = estrellas . significa () . round ( 2 ) # Calcula la distribución de estrellas
stars_histogram = stars . value_counts () . sort_index () stars_histogram / = float ( stars_histogram . sum ()) # Dibuja un gráfico de
barras que muestre la distribución de las calificaciones de estrellas ax = fig .
index + 1 ) stars_histogram . plot ( kind = "bar" , legend = None , color = "darkblue" ,
Ave: { average_stars }, Total : { total } ") plt . tight_layout () plt . mostrar ()

```

Nos hemos limitado por ciudad y categoría para centrarnos en los hoteles de Las Vegas. Ejecutamos ese código, obtenemos el gráfico en la [Figura 7-5](#). Tenga en cuenta que el eje x representa la calificación en estrellas del hotel y el eje y representa el porcentaje general de cada calificación.

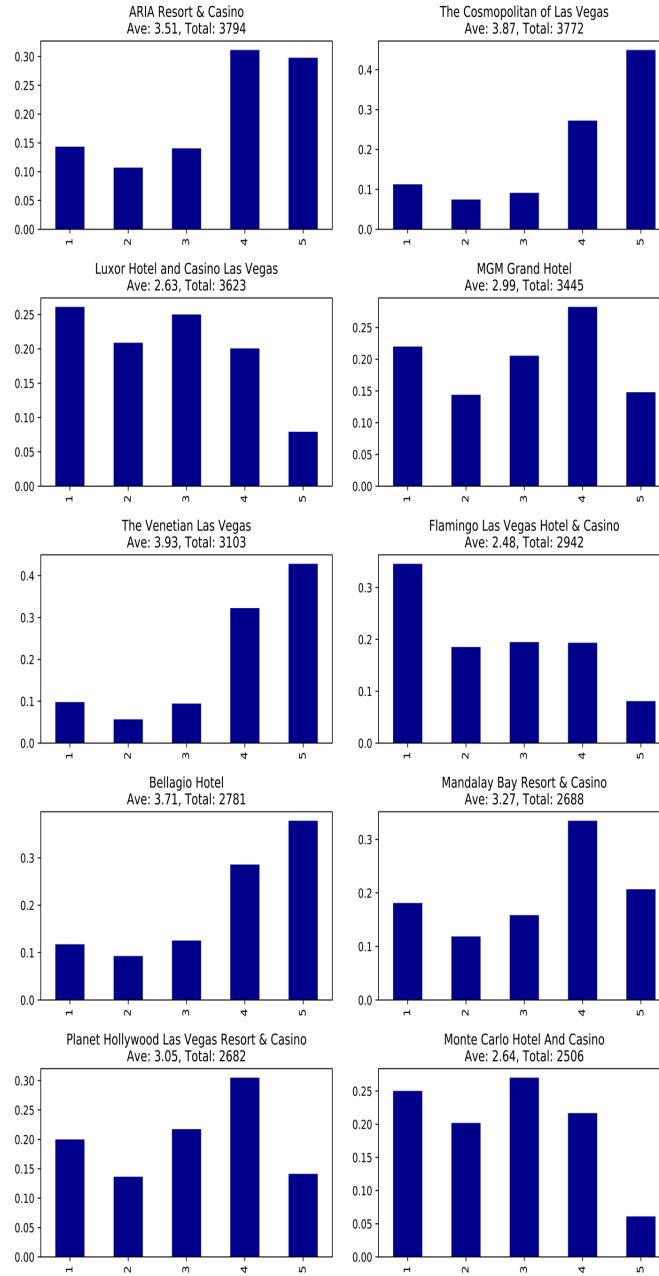


Figura 7-5. Los 10 hoteles más revisados, con el número de estrellas en el eje x y su porcentaje de calificación general en el eje y

Estos hoteles tienen muchas reseñas, más de lo que cualquiera podría leer. Sería mejor mostrar a nuestros usuarios el contenido de las revisiones más relevantes y hacerlos más prominentes en nuestra aplicación. Para realizar este análisis, pasaremos de la exploración de gráficos básicos al uso de algoritmos de gráficos.

## Encontrando críticos de hoteles influyentes

Una forma en la que podemos decidir qué revisiones publicar son mediante el pedido de revisiones basadas en la influencia del revisor en Yelp. Nosotros correremos el algoritmo de PageRank sobre el gráfico proyectado de todos los usuarios que han revisado al menos tres hoteles. Recuerde de capítulos anteriores que

una proyección puede ayudar a filtrar información no esencial, así como a agregar datos de relaciones (a veces inferidos). Usaremos el gráfico de amigos de Yelp (introducido en la ["Red social de Yelp"](#)) como las relaciones entre los usuarios. El algoritmo de PageRank descubrirá a aquellos revisores con más influencia sobre más usuarios, incluso si no son amigos directos.

### NOTA

Si dos personas son amigas de Yelp, hay dos relaciones de AMIGOS entre ellas. Por ejemplo, si A y B son amigos, habrá una relación de AMIGOS de A a B y otra de B a A.

Necesitamos escribir una consulta que proyecte un subgrafo de usuarios con más de tres revisiones y luego ejecute el algoritmo PageRank sobre ese subgrafo proyectado.

Es más fácil comprender cómo funciona la proyección de subgrafo con un pequeño ejemplo. [La figura 7-6](#) muestra una gráfica de tres amigos comunes: Mark, Arya y Praveena. Mark y Praveena han revisado tres hoteles y formarán parte del gráfico proyectado. Arya, por otro lado, solo ha revisado un hotel y, por lo tanto, será excluido de la proyección.

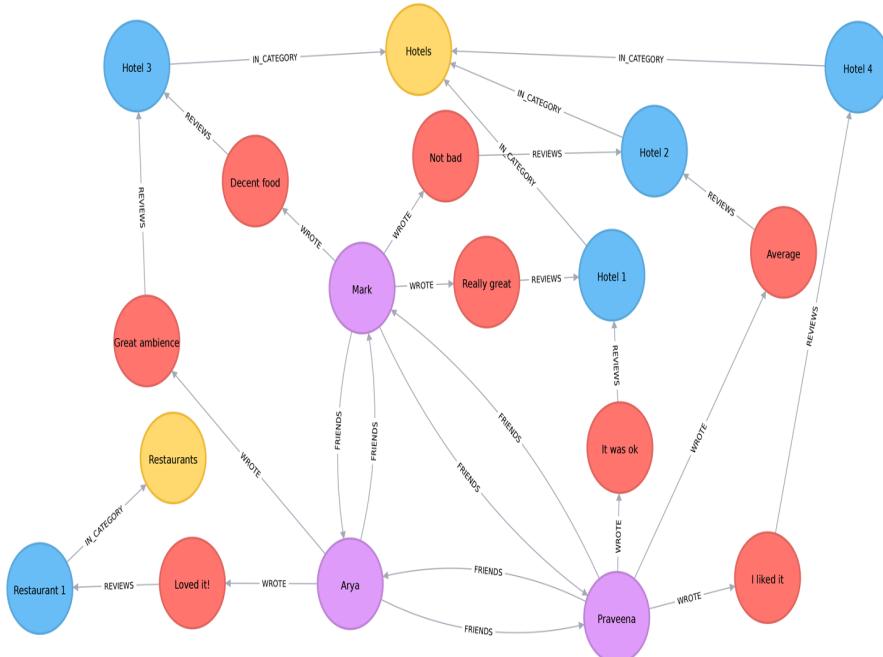


Figura 7-6. Un ejemplo de gráfico de Yelp

Nuestro gráfico proyectado solo incluirá a Mark y Praveena, como se muestra en la [Figura 7-7](#).

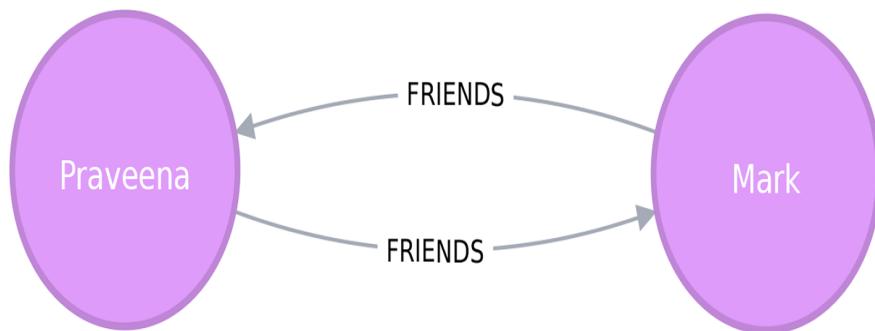


Figura 7-7. Nuestra muestra de gráfico proyectado.

Ahora que hemos visto cómo funcionan las proyecciones de gráficos, avancemos. La siguiente consulta ejecuta el algoritmo PageRank sobre nuestro gráfico proyectado y almacena el resultado en la propiedad hotelPageRank en cada nodo:

```
CALL algo.pageRank ('MATCH (u: User) - [: WROTE] -> () - [: REVIEWS] -> () - [: IN_CATEGORY] -> (: Category {name: $category}) WITH u, count (*) AS comentarios WHERE comentarios >= $cutOff RETURN id (u) AS id ', ' MATCH (u1: User) - [: WROTE] -> () - [: REVIEWS] -> () - [: IN_CATEGORY ] -> (: Categoría {nombre: $categoría}) MATCH (u1) - [: AMIGOS] -> (u2) RETURN id (u1) AS source, id (u2) AS target ', {graph: "cypher" , write: true , writeProperty: "hotelPageRank" , params: {category: "Hotels" , cutOff: 3}} )
```

Es posible que haya notado que no establecimos el factor de amortiguamiento o el límite de iteración máximo discutido en el [Capítulo 5](#). Si no se establece explícitamente, Neo4j se establece de manera predeterminada en un factor de amortiguamiento de 0.85 con maxIterations establecido en 20`.

Ahora veamos la distribución de los valores de PageRank para saber cómo filtrar nuestros datos:

```
MATCH (u: User) DONDE ESTÁ (u.hotelPageRank) RETURN count (u.hotelPageRank) AS count , avg (u.hotelPageRank) AS ave, percentileDisc (u.hotelPageRank, 0.5) AS `50%` , percentileDisc (u.hotelPageRank, 0.75) AS `75%` , percentileDisc (u.hotelPageRank, 0.90) AS `90%` , percentileDisc (u.hotelPageRank, 0.95) AS `95%` , percentileDisc (u.hotelPageRank, 0.99) AS `99%` , percentileDisc (u.hotelPageRank, 0.999) AS ``99.9%, percentileDisc (u.hotelPageRank, 0.9999) AS `99.99%` , percentileDisc (u.hotelPageRank, 0.99999) AS `99.999%` , percentileDisc (u.hotelPageRank, 1) AS `100%`
```

Si ejecutamos esa consulta obtendremos esta salida:

contar	Cra	50%	75%	90%	95%	99%	99.9%	99.99%	99.999%	100%
1326101	0.1614898	0.15	0.15	0.157497	0.181875	0.330081	1.649511	6.825738	15.27376	22.98046

Para interpretar esta tabla de percentiles, el valor del 90% de 0.157497 significa que el 90% de los usuarios tuvieron una puntuación de PageRank más baja. El 99.99% refleja el rango de influencia para los primeros 0.0001% de los revisores y el 100% es simplemente la puntuación más alta de PageRank.

Es interesante que el 90% de nuestros usuarios tengan una puntuación inferior a 0,16, que está cerca del promedio general, y solo un poco más que los 0,15 con los que se inicializa mediante el algoritmo PageRank. Parece que estos datos reflejan una distribución de ley de poder con algunos revisores muy influyentes.

Debido a que estamos interesados en encontrar solo a los usuarios más influyentes, escribiremos una consulta que solo encuentre usuarios con un puntaje de PageRank en el 0.001% más alto de todos los usuarios. La siguiente consulta encuentra revisores con un puntaje de PageRank superior a 1.64951 (note que es el grupo del 99.9%):

```
// Sólo encontrar usuarios que tienen una cuenta en la parte superior hotelPageRank 0,001% de los usuarios PARTIDO (u: usuario) DONDE u.hotelPageRank > 1,64951 // Encuentra el top 10 de los usuarios CON u ORDER BY u.hotelPageRank DESC LIMIT 10 RETORNO u .name AS name, u.hotelPageRank AS pageRank, tamaño ((u) - [: WROTE] -> () - [: REVIEWS] -> () - [: IN_CATEGORY] -> (: Category {name: "Hotels" }) AS HotelReviews, tamaño ((u) - [: WROTE] -> ()) AS totalReviews, tamaño ((u) - [: FRIENDS] - () ) AS friends
```

Si ejecutamos esa consulta obtendremos los resultados vistos aquí:

nombre	rango de página	opinión del hotel	Revisiones totales	amigos
Phil	17.361242	15	134	8154
Felipe	16.871013	21	620	9634
Villancico	12.416060999999997	6	119	6218
Misti	12.239516000000004	19	730	6230
José	12.003887499999998	5	32	6596
Miguel	11.460049	13	51	6572
J	11.431505999999997	103	1322	6498
Abby	11.376136999999998	9	82	7922
Erica	10.993773	6	15	7071
Cachondo	10.748785999999999	21	125	7846

Estos resultados nos muestran que Phil es el crítico más creíble, aunque no ha revisado muchos hoteles. Probablemente esté conectado con algunas personas muy influyentes, pero si quisieramos un flujo de nuevas reseñas, su perfil no sería la mejor opción. Philip tiene un puntaje ligeramente más bajo, pero tiene la mayoría de los amigos y ha escrito cinco veces más críticas que Phil. Si bien J ha escrito el mayor número de comentarios de todos y tiene un número razonable de amigos, el puntaje de PageRank de J no es el más alto, pero aún está entre los 10 mejores. Para nuestra aplicación, elegimos resaltar los comentarios de hoteles de Phil, Philip y J para darnos la mezcla correcta de influencers y número de revisiones.

Ahora que hemos mejorado nuestras recomendaciones en la aplicación con revisiones relevantes, pasemos al otro lado de nuestro negocio: la consultoría.

## Consultoría de viajes de negocios

Como parte de nuestro servicio de consultoría, los hoteles se suscriben para recibir una alerta cuando un visitante influyente escribe sobre su estadía para que puedan tomar las medidas necesarias. Primero, veremos las calificaciones de Bellagio, ordenadas por los revisores más influyentes:

```
query = "" "\MATCH (b: Business {name: $ hotel}) MATCH (b) <- [:REVIEWS] - (review) <- [:WROTE] - (user) DONDE existe
(user.hotelPageRank) RETURN user.name AS name, user.hotelPageRank AS pageRank, review.stars AS stars "" " con el controlador .
sesión () como sesión : params = { "hotel" : "Hotel Bellagio" } df = pd . DataFrame ([ dict ( record ) para grabar en sesión .
ejecutar ( consulta , params )]) df = df . round ( 2 ) df = df [[ "name" , "pageRank" , "stars" ]] top_reviews = df . sort_values ( by = [ "pageRank" ] , ascendente = False ) . encabezado ( 10 ) de impresión ( tabular (la opción más alta , encabezados = 'keys' ,
tablefmt = 'psql' , showindex = False ))
```

Si ejecutamos ese código obtendremos esta salida:

nombre	rango de página	estrellas
Misti	12.239516000000004	5
Miguel	11.460049	4
J	11.431505999999997	5
Erica	10.993773	4
Christine	10.740770499999998	4
Jeremy	9.576763499999998	5
Connie	9.118103499999998	5
Joyce	7.621449000000001	4
Enrique	7.299146	5
Flora	6.7570075	4

Tenga en cuenta que estos resultados son diferentes de nuestra tabla anterior de los mejores revisores de hoteles. Eso es porque aquí solo estamos mirando a los críticos que han calificado al Bellagio.

Las cosas se ven bien para el equipo de servicio al cliente del hotel en el Bellagio: los 10 principales influenciadores le dan a su hotel una buena clasificación. Es posible que quieran animar a estas personas a visitar nuevamente y compartir sus experiencias.

¿Hay algún invitado influyente que no haya tenido una experiencia tan buena? Podemos ejecutar el siguiente código para encontrar a los huéspedes con el PageRank más alto que calificó su experiencia con menos de cuatro estrellas:

```
query = "" "\MATCH (b: Business {name: $ hotel}) MATCH (b) <- [:REVIEWS] - (review) <- [:WROTE] - (user) DONDE existe
(user.hotelPageRank) AND review.stars <$ goodRating RETURN user.name AS name, user.hotelPageRank AS pageRank, review.stars
AS estrellas "" " con el controlador . session () as sesión : params = { "hotel" : "Bellagio Hotel" , "goodRating" : 4 } df = pd .
) para grabar en sesión . ejecutar ( consulta , params )]) df = df . round ( 2 ) df = df [[ "name" , "pageRank" , "stars" ]]
top_reviews = df . sort_values ( by = [ "pageRank" ] , ascendente = False ) . cabeza ( 10 ) imprimir ( tabular ( topning , headers
= 'keys' , tablefmt = 'psql' , showindex = False )
```

Si ejecutamos ese código obtendremos los siguientes resultados:

nombre	rango de página	estrellas
Chris	5.84	3
Lorrie	4.95	2
Dani	3.47	1
Víctor	3.35	3
Francine	2.93	3

Rex	2.79	2
Jon	2.55	3
Rachel	2.47	3
Leslie	2.46	2
Benay	2.46	3

Chris y Lorrie, nuestros usuarios mejor calificados que dan calificaciones más bajas a Bellagio, se encuentran entre los 1,000 usuarios más influyentes (según los resultados de nuestra consulta anterior), por lo que quizás se justifique un alcance personal. Además, como muchos revisores escriben durante su estadía, las alertas en tiempo real sobre personas influyentes pueden facilitar interacciones aún más positivas.

### Bellagio cruz-promoción

Después de que los ayudamos a encontrar críticos influyentes, Bellagio nos ha pedido que ayudemos a identificar otras empresas para realizar promociones cruzadas con la ayuda de clientes bien conectados. En nuestro escenario, recomendamos que aumenten su base de clientes al atraer a nuevos invitados de diferentes tipos de comunidades como una nueva oportunidad. Podemos usar el algoritmo de centralidad de intermediación que analizamos anteriormente para determinar qué revisores de Bellagio no solo están bien conectados en toda la red de Yelp, sino que también pueden actuar como un puente entre diferentes grupos.

Solo estamos interesados en encontrar personas influyentes en Las Vegas, así que primero etiquetaremos a esos usuarios:

```
PARTIDO (u: Usuario) DONDE ESTÁ ((u) - [: WROTE] -> () - [: REVIEWS] -> () - [: IN_CITY] -> (: City {name: "Las Vegas" }))  
SET u: LasVegas
```

Tomaría mucho tiempo ejecutar el algoritmo de centralidad de intermediación sobre nuestros usuarios de Las Vegas, así que en su lugar usaremos la variante RA-Brandes. Este algoritmo calcula una puntuación de intermediación mediante el muestreo de nodos y solo explora los caminos más cortos hasta una cierta profundidad.

Después de algunos experimentos, mejoramos los resultados con unos pocos parámetros establecidos de manera diferente a los valores predeterminados. Usaremos rutas más cortas de hasta 4 saltos (máx. Profundidad de 4) y muestrearemos el 20% de los nodos (probabilidad de 0.2). Tenga en cuenta que aumentar el número de saltos y nodos generalmente aumentará la precisión, pero a costa de más tiempo para calcular los resultados. Para cualquier problema particular, los parámetros óptimos generalmente requieren pruebas para identificar el punto de rendimientos decrecientes.

La siguiente consulta ejecutará el algoritmo y almacenará el resultado en la propiedad between :

```
CALL algo.betweenness.sampled ('LasVegas' , 'FRIENDS' , {write: true , writeProperty: "between" , maxDepth: 4, probabilidad: 0.2} )
```

Antes de usar estos puntajes en nuestras consultas, escribamos una consulta exploratoria rápida para ver cómo se distribuyen los puntajes:

```
MATCH (u: User) DONDE ESTÁ (u.between) RETURN count (u.between) AS count , avg (u.between) AS ave, toInteger  
(percentileDisc (u.between, 0.5)) AS `50%` , toInteger (percentileDisc (u.between, 0.75)) AS `75%` , toInteger (percentileDisc  
(u.between, 0.90)) AS `90%` , toInteger (percentileDisc (u.between, 0.95)) AS `95%` , toInteger (percentileDisc (u.between, 0.99)) AS  
`99%` , toInteger (percentileDisc (u.between, 0.999)) AS `99.9%` , toInteger (percentileDisc (u.between, 0.9999)) AS `99.99%` , toInteger (percentileDisc  
(u.between, 0.99999)) AS `99.999%` , toInteger (percentileDisc (u.between, 0.999999)) AS `99.9999%` , toInteger (percentileDisc  
(u.between, 1)) AS p100
```

Si ejecutamos esa consulta veremos el siguiente resultado:

contar	Cra	50%	75%	90%	95%	99%	99.9%	99.99%	99.999%	100%
506028	320538.6014 0	10005	318944	1001655	4436409	34854988	214080923	621434012	1998032952	

La mitad de nuestros usuarios tienen una puntuación de 0, lo que significa que no están bien conectados. El percentil 1 (columna del 99%) se encuentra en al menos 4 millones de rutas más cortas entre nuestro conjunto

de 500,000 usuarios. Considerados juntos, sabemos que la mayoría de nuestros usuarios están mal conectados, pero algunos ejercen un gran control sobre la información; Este es un comportamiento clásico de las redes del mundo pequeño.

Podemos averiguar quiénes son nuestros superconectores ejecutando la siguiente consulta:

```
MATCH (u: User) - [: WROTE] -> () - [: REVIEWS] -> (: Business {name: "Bellagio Hotel" }) DONDE existe (u.between) RETURN
u.name AS user, toInteger ( u.between) AS betweenness, u.hotelPageRank AS pageRank, tamaño ((u) - [: WROTE] -> () - [: 
REVIEWS] -> () - [: IN_CATEGORY] -> (: Category {name: "Hoteles" })) AS hotelReviews ORDER BY u.between DESC LIMIT
10
```

La salida es la siguiente:

<b>usuario</b>	<b>entreidad</b>	<b>rango de página</b>	<b>opinión del hotel</b>
Misti	841707563	12.239516000000004	19
Christine	236269693	10.740770499999998	dieciséis
Erica	235806844	10.993773	6
Micro	215534452	NULO	2
J	192155233	11.431505999999997	103
Miguel	161335816	5.105143	31
Jeremy	160312436	9.576763499999998	6
Miguel	139960910	11.460049	13
Chris	136697785	5.838922499999999	5
Connie	133372418	9.118103499999998	7

Vemos a algunas de las mismas personas que vimos anteriormente en nuestra consulta de PageRank, con Mike como una excepción interesante. Fue excluido de ese cálculo porque no ha revisado suficientes hoteles (tres fue el límite), pero parece que está bastante bien conectado en el mundo de los usuarios de Yelp de Las Vegas.

En un esfuerzo por llegar a una variedad más amplia de clientes, veremos otras preferencias que muestran estos "conectores" para ver qué debemos promover. Muchos de estos usuarios también han revisado restaurantes, por lo que escribimos la siguiente consulta para averiguar cuáles les gustan más:

```
// Encuentre los 50 usuarios principales que han revisado el MATCH de Bellagio (u: Usuario) - [: WROTE] -> () - [: REVISIONES] ->
(: Empresa {nombre: "Bellagio Hotel" }) DÓNDE se encuentra entre. > 4436409 CON u ORDER BY u.between LIMIT DESC 50 // 
Encuentra los restaurantes aquellos usuarios han revisado en las Vegas PARTIDO (u) - [: ESCRIBIÓ] -> (opinión) - [: Comentarios] - 
(negocio) DONDE (negocio) - [: IN_CATEGORY] -> (: Category {name: "Restaurants" }) AND (business) - [: IN_CITY] -> (: City 
{name: "Las Vegas" }) // Solo incluye restaurantes que tengan más de 3 opiniones de estos usuarios CON business, avg 
(review.stars) AS averageReview, count (*) AS numberofReviews WHERE numberofReviews > = 3 RETURN business.name AS 
business, averageReview, numberofReviews ORDEN POR averageReview DESC , numberofReviews DESC LIMIT 10
```

Esta consulta encuentra nuestros 50 conectores más influyentes, y encuentra los 10 mejores restaurantes de Las Vegas donde al menos 3 de ellos han calificado al restaurante. Si lo ejecutamos, veremos la salida que se muestra aquí:

<b>negocio</b>	<b>promedioRevisión</b>	<b>numberOfReviews</b>
Jean Georges Steakhouse	5.0	6
Sushi House Goyemon	5.0	6
Arte de los sabores	5.0	4
é de José Andrés	5.0	4
Parma por el chef marc	5.0	4
Yonaka Japonés Moderno	5.0	4
Kabuto	5.0	4

Cosecha de Roy Ellamar	5.0	3
Portofino por el chef Michael LaPlaca	5.0	3
Eateria de Montesano	5.0	3

Ahora podemos recomendar que Bellagio realice una promoción conjunta con estos restaurantes para atraer a nuevos huéspedes de grupos a los que normalmente no pueden llegar. Los superconectores que califican bien al Bellagio se convierten en nuestro proxy para estimar qué restaurantes pueden captar la atención de los nuevos tipos de visitantes objetivo.

Ahora que hemos ayudado a Bellagio a llegar a nuevos grupos, vamos a ver cómo podemos usar la detección de la comunidad para mejorar aún más nuestra aplicación..

## Encontrar categorías similares

Si bien nuestros usuarios finales utilizan la aplicación para buscar hoteles, queremos mostrar otras empresas en las que podrían estar interesados. El conjunto de datos de Yelp contiene más de 1,000 categorías, y es probable que algunas de esas categorías sean similares entre sí. Usaremos esa similitud para hacer recomendaciones en la aplicación para nuevas empresas que nuestros usuarios probablemente encuentren interesantes.

Nuestro modelo de gráfico no tiene ninguna relación entre categorías, pero podemos usar las ideas descritas en "[Gráficos monopartidos, bipartidos y partidarios k](#)" para construir un gráfico de similitud de categorías basado en cómo se clasifican las empresas.

Por ejemplo, imagine que solo una empresa se clasifica a sí misma tanto en Hoteles como en Recorridos históricos, como se ve en la [Figura 7-8](#) .

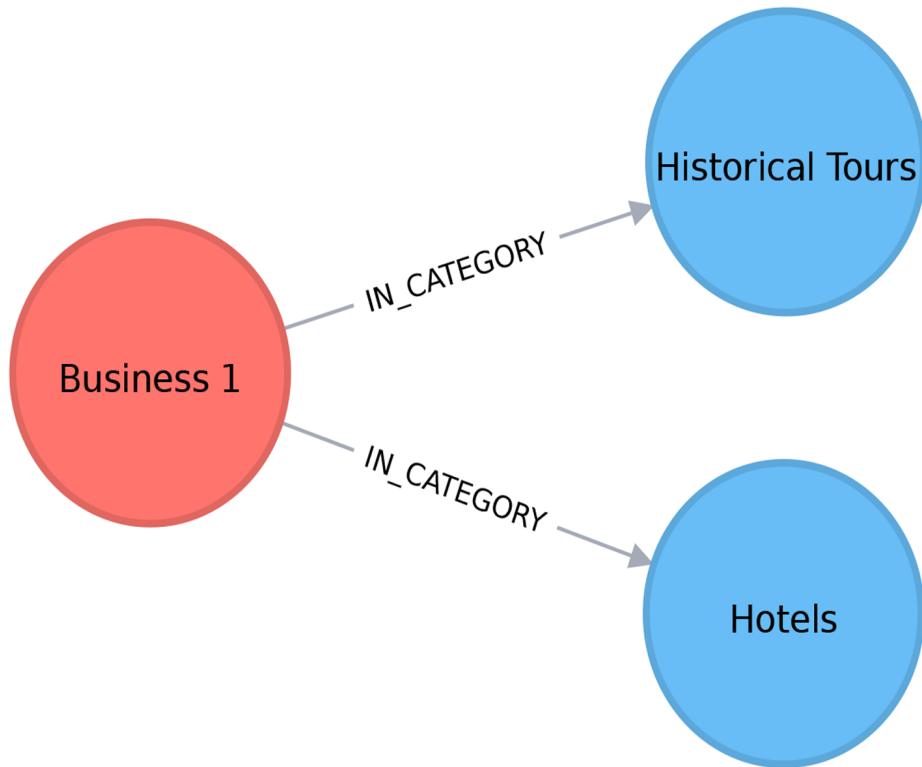


Figura 7-8. Un negocio con dos categorías.

Esto daría como resultado un gráfico proyectado que tiene un enlace entre Hoteles y Tours Históricos con un peso de 1, como se ve en la [Figura 7-9](#) .

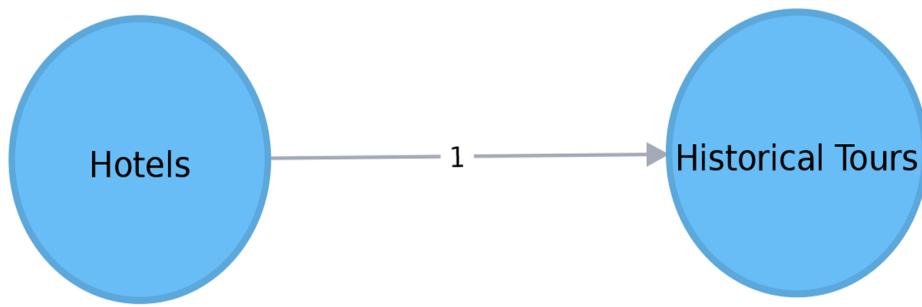


Figura 7-9. Un gráfico de categorías proyectadas.

En este caso, no tenemos que crear el gráfico de similitud; en cambio, podemos ejecutar un algoritmo de detección de la comunidad como la propagación de etiquetas en un gráfico de similitud proyectado. El uso de la propagación de etiquetas agrupará las empresas de manera efectiva en torno a la supercategoría con la que tienen más en común:

```

CALL algo.labelPropagation.stream ('MATCH (c: Category) RETURN id (c) AS id', 'MATCH (c1: Category) <- [: IN_CATEGORY] - () - [: IN_CATEGORY] -> (c2: Category ) WHERE id (c1) <id (c2) RETURN id (c1) AS source, id (c2) AS target, count (*) AS weight ', {graph: "cypher" } ) YIELD nodeId, label MATCH (c: Categoría) WHERE id (c) = nodeId MERGE (sc: SuperCategory {name: "SuperCategory-" + label}) MERGE (c) - [: IN_SUPER_CATEGORY] -> (sc)
  
```

Démosle a esas supercategorías un nombre más amigable, el nombre de su categoría más grande funciona bien aquí:

```

MATCH (sc: SuperCategory) <- [: IN_SUPER_CATEGORY] - (category) WITH sc, category, size ((category) <- [: IN_CATEGORY] - () ) como tamaño ORDER BY tamaño DESC WITH sc, collect (category.name ) [0] as bigCategory SET sc.friendlyName = "SuperCat" + biggestCategory
  
```

Podemos ver una muestra de categorías y supercategorías en la [Figura 7-10](#).

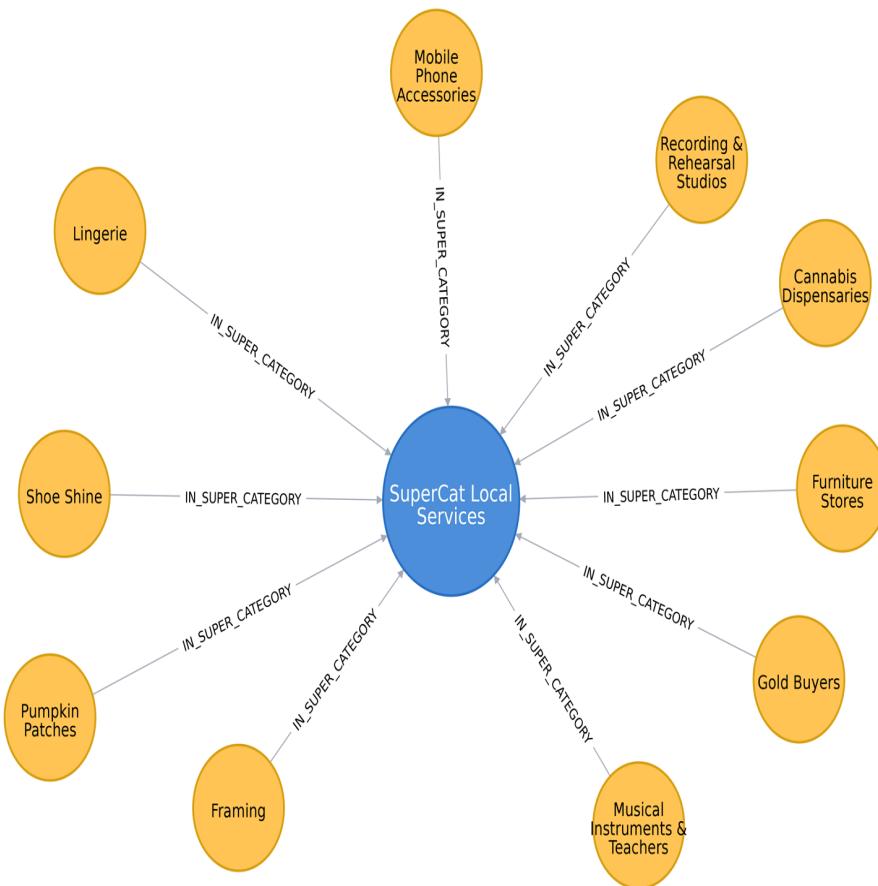


Figura 7-10. Categorías y supercategorías

La siguiente consulta encuentra las categorías similares más comunes a los hoteles en Las Vegas:

```

MATCH (hoteles: Categoría {nombre: "Hoteles" }), (lasVegas: Ciudad {nombre: "Las Vegas" }), (hoteles) - [:IN_SUPER_CATEGORY] -> () <- [:IN_SUPER_CATEGORY] - (otherCategory) RETORNO otherCategory.name AS otherCategory, tamaño ((otherCategory) <- [:IN_CATEGORY] - (: Negocios) - [:IN_CITY] -> (lasVegas)) AS empresas ORDER BY recuento DESC LIMIT 10

```

Si ejecutamos esa consulta veremos el siguiente resultado:

otra categoría	negocios
Excursiones	189
Alquiler de coches	160
Limusinas	84
Resorts	73
Traslados al aeropuerto	52
Taxis	35
Alquileres de vacaciones	29
Aeropuertos	25
aerolíneas	23
Alquiler de motos	19

¿Estos resultados parecen extraños? Obviamente, los taxis y las excursiones no son hoteles, pero recuerde que esto se basa en categorizaciones autoinformadas. Lo que el algoritmo de propagación de etiquetas realmente nos muestra en este grupo de similitud son los negocios y servicios adyacentes.

Ahora encontremos algunos negocios con una calificación superior a la media en cada una de esas categorías:

```

// Encuentra negocios en Las Vegas que tengan la misma Supercategoría que Hoteles MATCH (hoteles: Categoría {nombre: "Hoteles" }), (hoteles) - [:IN_SUPER_CATEGORY] -> () <- [:IN_SUPER_CATEGORY] - (otherCategory) , (otherCategory) <- [:IN_CATEGORY] - (business) WHERE (business) - [:IN_CITY] -> (: City {name: "Las Vegas" }) // Seleccione 10 categorías al azar y calcule la clasificación de estrellas del percentil 90 CON otherCategory, count (*) AS count , collect (business) AS empresas, percentileDisc (business.averageStars, 0.9) AS p90Stars ORDER BY rand () DESC LIMIT 10 // Seleccione negocios de cada una de esas categorías que tengan una calificación promedio // más alta que el percentil 90 utilizando un patrón de comprensión CON otra categoría, [b en negocios donde b.averageStars > = p90Stars] AS business // Seleccione un negocio por categoría CON otra categoría , negocios [toInteger (rand () * tamaño (negocios))] COMO negocio REGRESAR otherCategory.name AS otherCategory, business.name AS business, business.averageStars AS averageStars

```

En esta consulta usamos [la comprensión de patrones](#) por primera vez. La comprensión de patrones es una construcción de sintaxis para crear una lista basada en la coincidencia de patrones. Encuentra un patrón específico usando una cláusula MATCH con una cláusula WHERE para predicados y luego produce una proyección personalizada. Esta característica de Cypher se agregó basándose en la inspiración de [GraphQL](#) , un lenguaje de consulta para las API.

Si ejecutamos esa consulta vemos el siguiente resultado:

otra categoría	negocio	estrellas promedio
Alquiler de motos	Adrenaline Rush Slingshot Rentals	5.0
Bucear	Sin City Scuba	5.0
Casas de huéspedes	Hotel Del Kacvinsky	5.0
Alquiler de coches	El equipo líder	5.0
Tours de comida	Taste BUZZ Food Tours	5.0
Aeropuertos	Soporte de vuelo de firma	5.0
Transporte público	JetSuiteX	4.6875
Estaciones de esquí	Trikke Las Vegas	4.83333333333332
Servicio de coche de ciudad	MW Travel Vegas	4.866666666666665

Campamentos	McWilliams Campground	3.875
-------------	-----------------------	-------

Luego podemos hacer recomendaciones en tiempo real basadas en el comportamiento inmediato de la aplicación del usuario. Por ejemplo, mientras los usuarios buscan en los hoteles de Las Vegas, ahora podemos destacar una variedad de empresas adyacentes de Las Vegas con buenas calificaciones. Podemos generalizar estos enfoques a cualquier categoría de negocios, como restaurantes o teatros, en cualquier ubicación.

### EJERCICIOS PARA LECTORES

- ¿Puede trazar cómo varían las revisiones de los hoteles de una ciudad a lo largo del tiempo?
- ¿Qué pasa con un hotel en particular u otro negocio?
- ¿Hay alguna tendencia (estacional o de otro tipo) en popularidad?
- ¿Los revisores más influyentes se conectan (out-link) solo a otros revisores influyentes?

## Analizando los datos de vuelo de las aerolíneas con Apache Spark

En esta sección, usaremos un escenario diferente para ilustrar el análisis de los datos de aeropuertos de Estados Unidos con Spark. Imagina que eres un científico de datos con un calendario de viajes considerable y te gustaría obtener información sobre los vuelos y retrasos de las aerolíneas. Primero exploraremos la información de aeropuertos y vuelos y luego analizaremos más a fondo los retrasos en dos aeropuertos específicos. La detección de la comunidad se utilizará para analizar rutas y encontrar el mejor uso de nuestros puntos de viajero frecuente.

La Oficina de Estadísticas de Transporte de los Estados Unidos pone a disposición una [cantidad significativa de información sobre el transporte](#). Para nuestro análisis, usaremos sus datos de desempeño a tiempo de los viajes aéreos de mayo de 2018, que incluyen los vuelos que se originan y terminan en los Estados Unidos en ese mes. Para agregar más detalles sobre los aeropuertos, como la información de ubicación, también [cargaremos](#) datos de una fuente separada, [OpenFlights](#).

Vamos a cargar los datos en Spark. Al igual que en las secciones anteriores, nuestros datos están en archivos CSV que están disponibles en el [repositorio Github del libro](#).

```
nodos = chispa . lea . csv ( "data / airports.csv" , header = False ) cleaner_nodes = ( nodes . select ( "_c1" , "_c3" , "_c4" , "_c6" , "_c7" ) . filter ( "_c3 = 'United States'" ) . withColumnRenamed ( "_c1" , "nombre" ) . withColumnRenamed ( " ) s . withColumnRenamed ( "_c6" , "latitude" ) . withColumnRenamed ( "_c7" , "longitud" ) . drop ( "_c3" ) ) cleaner_nodes = cleaner_nodes [ cleaner_nodes [ "id" ] != "N" ] relaciones = chispa . lea . csv ( "data / 188591317_T_ONTIME.csv" , "dst" ) . withColumnRenamed ( "DEP_DELAY" , "deptDelay" ) . withColumnRenamed ( "ARR_DELAY" , "arrDelay" ) . withColumnRenamed ( "FL_NUM" , "flightNumber" ) . withColumnRenamed ( "CRS_DEP_TIME" , "time" ) . withColumnRenamed ( "DISTANCE" , "distance" ) . withColumnRenamed ( "UNIQUE_CARRIER" , "aerolínea" ) . withColumn ( "deptDelay" , F . col ( "deptDelay" ) . reparto ( FloatType () ) ) . withColumn ( "arrDelay" , F . col ( "arrDelay" ) . withColumn ( "arrivalTime" , F . col ( "arrivalTime" ) . reparto ( IntegerType () ) ) . withColumn ( "arrivalTime" , F . col ( "arrivalTime" ) . reparto ( IntegerType () ) ) g = GraphFrame ( cleaned_nodes , cleaned_relationships )
```

Tenemos que hacer una limpieza en los nodos porque algunos aeropuertos no tienen códigos de aeropuerto válidos. Daremos a las columnas nombres más descriptivos y convertiremos algunos elementos en tipos numéricos apropiados. También debemos asegurarnos de que tengamos las columnas denominadas id, dst y src, ya que esto es lo que espera la biblioteca GraphFrames de Spark.

También crearemos un DataFrame separado que asigna los códigos de las aerolíneas a los nombres de las aerolíneas. Usaremos esto más adelante en este capítulo:

```
airlines_reference = ( spark . read . csv ( "data / airlines.csv" ) . select ( "_c1" , "_c3" ) . withColumnRenamed ( "_c1" , "name" ) . withColumnRenamed ( "_c3" , "code" ) ) airlines_reference = airlines_reference [ airlines_reference [ "code" ] != "null" ]
```

## Análisis exploratorio

Comencemos con un análisis exploratorio para ver cómo se ven los datos.

Primero veamos cuantos aeropuertos tenemos:

```
g . vértices . contar ()
```

1435

¿Y cuántas conexiones tenemos entre estos aeropuertos?

```
g . bordes . contar ()
```

616529

## Aeropuertos populares

¿Qué aeropuertos tienen más vuelos de salida? Podemos calcular el número de vuelos salientes desde un aeropuerto utilizando el algoritmo de grado de centralidad:

```
airports_degree = g . outDegrees . withColumnRenamed ( "id" , "oId" ) full_airports_degree = ( airports_degree . join ( g . vertices , airports_degree . oId == g . vertices . id ) . sort ( "outDegree" , ascendente = False ) . select ( "id" , "nombre" , "outDegree" ) ) full_airports_degree . show ( n = 10 , truncate = False )
```

Si ejecutamos ese código veremos el siguiente resultado:

carné de identidad	nombre	fuera de grado
ATL	Aeropuerto Internacional Hartsfield Jackson de Atlanta	33837
ORD	Aeropuerto Internacional de Chicago O'Hare	28338
DFW	Aeropuerto Internacional de Dallas-Fort Worth	23765
CLT	Aeropuerto Internacional Charlotte Douglas	20251
GUARIDA	Aeropuerto Internacional de Denver	19836
FLOJO	Aeropuerto Internacional de Los Angeles	19059
PHX	Aeropuerto Internacional Phoenix Sky Harbor	15103
OFS	Aeropuerto Internacional de San Francisco	14934
LGA	Aeropuerto La Guardia	14709
IAH	Aeropuerto Intercontinental George Bush de Houston	14407

La mayoría de las grandes ciudades de los Estados Unidos aparecen en esta lista: Chicago, Atlanta, Los Ángeles y Nueva York tienen aeropuertos populares. También podemos crear una representación visual de los vuelos salientes utilizando el siguiente código:

```
PLT . estilo . use ( 'fivethirtyeight' ) ax = ( full_airports_degree . toPandas () . head ( 10 ) . plot ( kind = 'bar' , x = 'id' , y = 'outDegree' , legend = None ) ) ax . xaxis . set_label_text ( "" ) plt . xticks ( rotación = 45 ) plt . tight_layout () plt . mostrar ()
```

El gráfico resultante se puede ver en la [Figura 7-11](#).

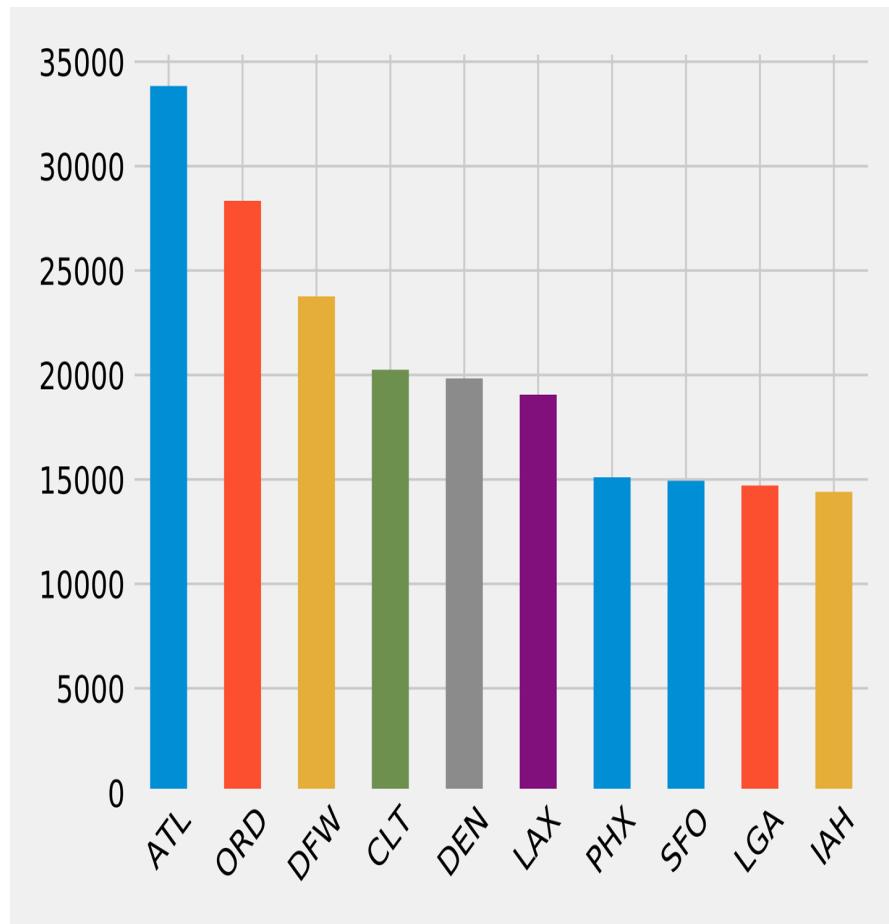


Figura 7-11. Vuelos salientes por aeropuerto

Es bastante sorprendente lo repentino que cae el número de vuelos. El Aeropuerto Internacional de Denver (DEN), el quinto aeropuerto más popular, tiene apenas la mitad de las peleas salientes que el Aeropuerto Internacional Hartsfield Jackson de Atlanta (ATL), en primer lugar.

## Retrasos de ORD

En nuestro escenario, con frecuencia viajamos entre las costas este y oeste y queremos ver retrasos en un punto intermedio como el Aeropuerto Internacional de Chicago O'Hare (ORD). Este conjunto de datos contiene datos de retardo de vuelo, por lo que podemos sumergirnos de inmediato.

El siguiente código encuentra el retraso promedio de los vuelos que salen de ORD agrupados por el aeropuerto de destino:

```
delayed_flights = ( g . bordes . filtrado ( "src = 'ORD' y deptDelay > 0" ) . GroupBy ( "dst" ) . agg ( F . avg ( "deptDelay" ), F . recuento ( "deptDelay" ) ) . withColumn ( "averageDelay" , F . ronda ( F . col ( "avg (deptDelay)" ), 2 ) ) . con Columna ( "NumberOfDelays" , F . Col ( "count (deptDelay)" )) ) ( delayed_flights . Unirse a ( g . Vértices , delayed_flights . Dst == g . Vértices . Id ) . Especie ( F . Desc ( "averageDelay" ) ) . select ( "dst" , "name" , "averageDelay" , "numberOfDelays" ) . espectáculo ( n = 10 , truncate = False ) )
```

Una vez que calculamos el retraso promedio agrupado por destino, unimos el marco de datos de Spark resultante con un DataFrame que contiene todos los vértices, de modo que podamos imprimir el nombre completo del aeropuerto de destino.

Ejecutar este código devuelve los 10 destinos con los peores retrasos:

dst	nombre	Retardo medio	numberOfDelays
CKB	Aeropuerto de North Central West Virginia	145.08	12
OGG	Aeropuerto de Kahului	119.67	9
MQT	Aeropuerto Internacional Sawyer	114.75	12
MULTITUD	Aeropuerto Regional de Mobile	102.2	10

TTN	Aeropuerto Trenton Mercer	101.18	17
AVL	Aeropuerto Regional de Asheville	98.5	28
ISP	Aeropuerto Long Island Mac Arthur	94.08	13
Congreso Nacional Africano	Aeropuerto Internacional Ted Stevens Anchorage	83.74	23
BTV	Aeropuerto Internacional de Burlington	83.2	25
CMX	Aeropuerto Memorial del Condado de Houghton	79.18	17

Esto es interesante, pero realmente destaca un punto de datos: ¡12 vuelos de ORD a CKB se retrasaron en más de 2 horas en promedio! Busquemos los vuelos entre esos aeropuertos y veamos qué está pasando:

```
from_expr = 'id = "ORD"' to_expr = 'id = "CKB"' ord_to_ckb = g . bfs ( from_expr , to_expr ) ord_to_ckb = ord_to_ckb . seleccionar ( F . col ( "e0.date" ), F . col ( "e0.time" ), F . col ( "e0.flightNumber" ), F . col ( "e0.deptDelay" ))
```

Luego podemos trazar los vuelos con el siguiente código:

```
ax = ( ord_to_ckb . sort ( "date" ) . toPandas () . plot ( kind = 'bar' , x = 'date' , y = 'deptDelay' , legend = None )) ax . xaxis . set_label_text ( "" ) plt . tight_layout () plt . mostrar ()
```

Si ejecutamos ese código, obtendremos el gráfico en la [Figura 7-12](#).

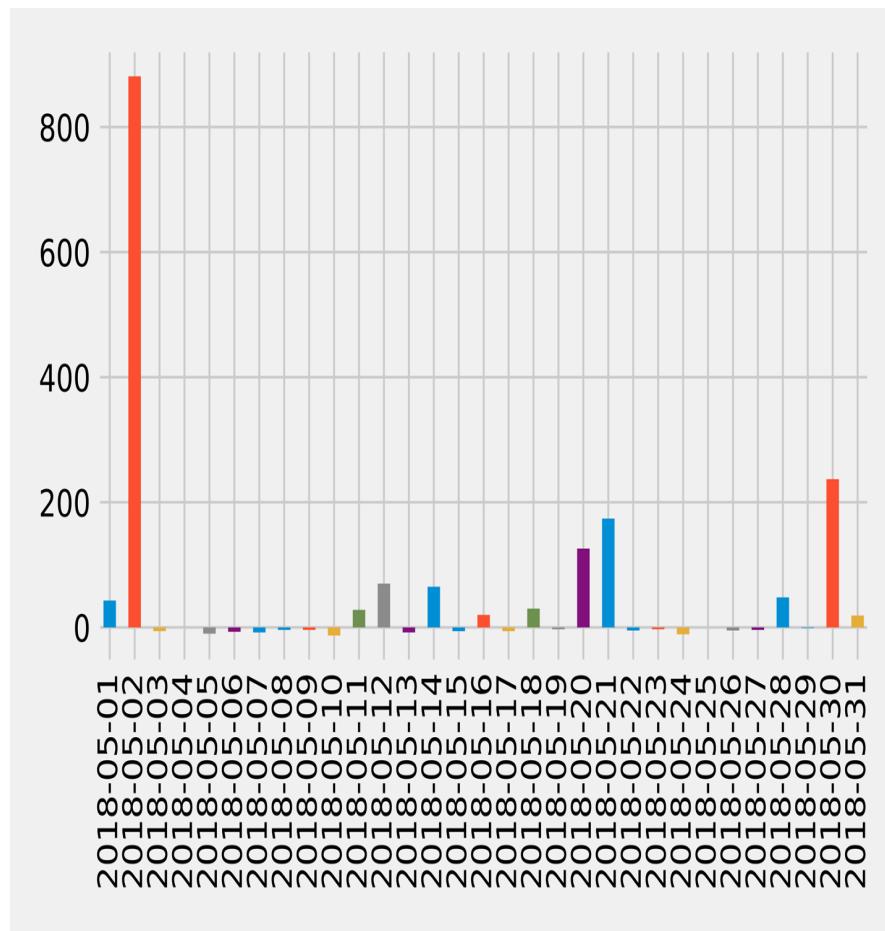


Figura 7-12. Vuelos desde ORD a CKB

Aproximadamente la mitad de los vuelos se retrasaron, pero la demora de más de 14 horas el 2 de mayo de 2018, ha sesgado masivamente el promedio.

¿Qué pasa si queremos encontrar retrasos que entran y salen de un aeropuerto costero? Esos aeropuertos a menudo se ven afectados por condiciones climáticas adversas, por lo que podríamos encontrar algunos retrasos interesantes.

## Mal día en la OFS

Consideremos las demoras en un aeropuerto conocido por problemas de “techo bajo” relacionados con la niebla: el Aeropuerto Internacional de San Francisco (SFO). Un método de análisis sería observar los motivos , que son subgrafos o patrones recurrentes.

### NOTA

El equivalente a los motivos en Neo4j son los patrones de gráficos, que se encuentran usando la cláusula MATCH o con expresiones de patrones en Cypher.

GraphFrames nos permite [buscar motivos](#) , por lo que podemos utilizar la estructura de vuelos como parte de una consulta. Usemos motivos para encontrar los vuelos más retrasados que entran y salen de la OFS el 11 de mayo de 2018. El siguiente código encontrará estos retrasos:

```
motivos = ( g . encontrar ( "(a) - [ab] -> (b); (b) - [bc] -> (c)" ) . filtro ( "" " (b.id = 'SFO') y (ab.date = '2018-05-11' y bc.date = '2018-05-11') y (ab.arrDelay > 30 o bc.deptDelay > 30) y (ab.flightNumber = bc.flightNumber) y (ab.airline = bc.airline) y (ab.time < bc.time) "" " ))
```

El motivo (a) - [ab] -> (b); (b) - [bc] -> (c) encuentra vuelos que entran y salen del mismo aeropuerto. Luego filtramos el patrón resultante para encontrar vuelos con:

- Una secuencia en la que el primer vuelo llega a la OFS y el segundo sale de la OFS
- Retrasos de más de 30 minutos al llegar o salir de la OFS
- El mismo número de vuelo y línea aérea.

Luego podemos tomar el resultado y seleccionar las columnas que nos interesan:

```
result = ( motivos . con Columna ( "delta" , motivos . bc . deptDelay - motivos . ab . arr . Delay ) . select ( "ab" , "bc" , "delta" ) . sort ( "delta" , ascendente = Falso ) ) consecuencia . seleccione ( F . col ( "ab.src" ) . alias ( "A1" ), F . col ( "ab.time" ) . alias ( "a1DeptTime" ), F . col ( "ab.arrDelay" ), F . col ( "ab.dst" ) . alias ( "A2" ), F . col ( "bc.time" ) . alias ( "a2DeptTime" ), F . col ( "bc.deptDelay" ), F .. alias ( "a3" ), F . col ( "ab.airline" ), F . col ( "ab.flightNumber" ), F . col ( "delta" ) ) . mostrar ()
```

También estamos calculando el delta entre los vuelos que llegan y los que salen para ver qué retrasos podemos atribuir realmente a la OFS.

Si ejecutamos este código obtendremos el siguiente resultado:

aerolínea	número de vuelo	a1	a1DeptTime	arrDelay	a2	a2DeptTime	deptDelay	a3	delta
WN	1454	PDX	1130	-18.0	OFS	1350	178.0	REBABA	196.0
OO	5700	ACV	1755	-9.0	OFS	2235	64.0	RDM	73.0
UA	753	BWI	700	-3.0	OFS	1125	49.0	DÍ	52.0
UA	1900	ATL	740	40.0	OFS	1110	77.0	SAN	37.0
WN	157	REBABA	1405	25.0	OFS	1600	39.0	PDX	14.0
DL	745	DTW	835	34.0	OFS	1135	44.0	DTW	10.0
WN	1783	GUARIDA	1830	25.0	OFS	2045	33.0	REBABA	8.0
WN	5789	PDX	1855	119.0	OFS	2120	117.0	GUARIDA	-2.0
WN	1585	REBABA	2025	31.0	OFS	2230	11.0	PHX	-20.0

El peor delincuente, WN 1454, se muestra en la fila superior; llegó temprano pero partió casi tres horas tarde. También podemos ver que hay algunos valores negativos en la columna arrDelay ; esto significa que el vuelo a la OFS fue temprano.

También tenga en cuenta que algunos vuelos, como el WN 5789 y el WN 1585, compensaron el tiempo en tierra en la OFS, como se muestra con un delta negativo.

## Aeropuertos interconectados por aerolínea

Ahora digamos que hemos viajado mucho, y esos puntos de viajero frecuente que estamos determinados a usar para ver tantos destinos tan eficientemente como sea posible expirarán pronto. Si partimos desde un

aeropuerto específico de EE. UU., ¿Cuántos aeropuertos diferentes podemos visitar y regresar al aeropuerto de inicio utilizando la misma aerolínea?

Primero, identifiquemos todas las aerolíneas y averigüemos cuántos vuelos hay en cada una de ellas:

```
Las aerolíneas = ( g . bordes . GroupBy ( "avión" ) . agg ( F . recuento ( "avión" ) . alias ( "vuelos" ) ) . ordenar ( "vuelos" , ascendente = False ) full_name_airlines = ( airlines_reference . unirse ( aerolíneas , airlines . airline == airlines_reference . code ) . seleccione ( "código" , "nombre" , "vuelos" ) )
```

Y ahora vamos a crear un gráfico de barras que muestre nuestras aerolíneas:

```
ax = ( full_name_airlines . toPandas () . plot ( kind = 'bar' , x = 'nombre' , y = 'vuelos' , leyenda = Ninguno ) ) ax . xaxis . set_label_text ( "" ) plt . tight_layout () plt . mostrar ()
```

Si ejecutamos esa consulta obtendremos la salida en la [Figura 7-13](#).

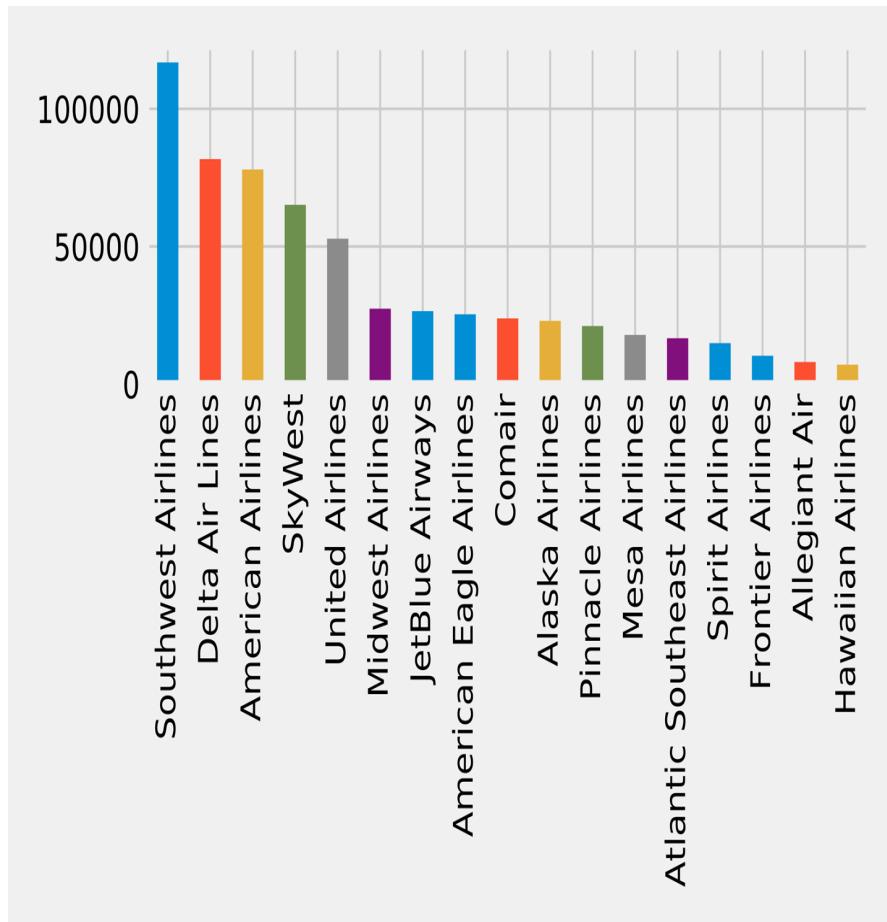


Figura 7-13. El número de vuelos por aerolínea.

Ahora, escribamos una función que use el algoritmo de componentes fuertemente conectados para encontrar agrupaciones de aeropuertos para cada aerolínea donde todos los aeropuertos tienen vuelos hacia y desde todos los demás aeropuertos de ese grupo:

```
def find_scc_components ( g , aerolínea ): # Crear un subgrafo que contenga solo vuelos en la aerolínea proporcionada
    airline_relationships = g . bordes [ g . bordes . airline == airline ] airline_graph = GraphFrame ( g . vértices , airline_relationships ) # Calcule los componentes fuertemente conectados scc = airline_graph . stronglyConnectedComponents ( maxiter = 10 )
    # Encontrar el tamaño del componente más grande y regresa esa retorno ( SCC . GroupBy ( "componente" ) . Agg ( F . Recuento ( "id" ) . Alias ( "tamaño" ) ) . Clase ( "tamaño" , ascendente = Falso ) . Toma ( 1 ) [ 0 ] [ "tamaño" ] )
```

Podemos escribir el siguiente código para crear un DataFrame que contenga cada aerolínea y la cantidad de aeropuertos de su componente más grande y fuertemente conectado:

```
# Calcule el componente con la mayor conexión para cada aerolínea airline_scc = [( aerolínea , find_scc_components ( g , aerolínea )) para aerolínea en aerolíneas . toPandas () [ "aerolínea" ] . tolist ()] airline_scc_df = spark . createDataFrame ( airline_scc , [ 'id' , 'sccCount' ] ) # Únase al SCC DataFrame con el DataFrame de la aerolínea para que podamos mostrar # el número de vuelos que tiene una aerolínea junto con el # de aeropuertos a los que se puede acceder en su componente más grande airline_reach = (
```

```
airline_scc_df . join ( full_name_airlines , full_name_airlines . code == airline_scc_df . id ) . select ( "code" , "name" , "flights" , "sccCount" ) . sort ( "sccCount" , ascendeante = Falso )
```

Y ahora vamos a crear un gráfico de barras que muestre nuestras aerolíneas:

```
ax = ( airline_reach . toPandas () . plot ( kind = 'bar' , x = 'name' , y = 'sccCount' , legend = None )) ax . xaxis . set_label_text ( "" ) plt . tight_layout () plt . mostrar ()
```

Si ejecutamos esa consulta obtendremos la salida en la [Figura 7-14](#).

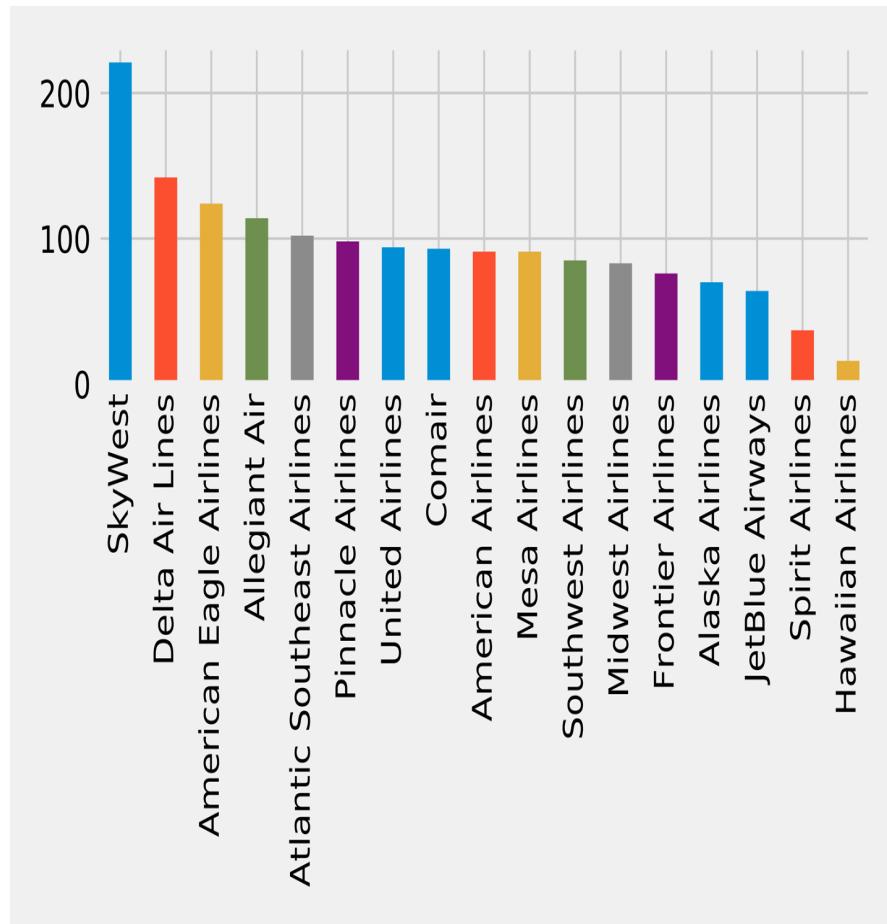


Figura 7-14. El número de aeropuertos accesibles por aerolínea.

SkyWest tiene la comunidad más grande, con más de 200 aeropuertos fuertemente conectados. Esto podría reflejar parcialmente su modelo comercial como una aerolínea afiliada que opera aeronaves utilizadas en vuelos para aerolíneas asociadas. Southwest, por otro lado, tiene el mayor número de vuelos, pero solo conecta alrededor de 80 aeropuertos.

Ahora digamos que la mayoría de los puntos de viajero frecuente que tenemos con Delta Airlines (DL). ¿Podemos encontrar aeropuertos que formen comunidades dentro de la red para esa aerolínea en particular?

```
líneas_aéreas_relaciones = g . bordes . filter ( "airline = 'DL'" ) airline_graph = GraphFrame ( g . vertices , airline_relationships ) clusters = airline_graph . labelPropagation ( maxiter = 10 ) ( racimos . especie ( "etiqueta" ) . GroupBy ( "etiqueta" ) . agg ( F . collect_list ( "id" ) . alias ( "aeropuertos" ) , f . cuenta ( "id" ) . alias ( "contar" ) ) . ordenar ( "contar" , ascendeante = Falso ) . show ( truncado = 70 , n = 10 ) )
```

Si ejecutamos esa consulta veremos el siguiente resultado:

etiqueta	aeropuertos	contar
1606317768706	[IND, ORF, ATW, RIC, TRI, XNA, ECP, AVL, JAX, SYR, BHM, GSO, MEM, C...]	89
1219770712067	[GEG, SLC, DTW, LAS, MAR, BOS, MSN, SNA, JFK, TVC, LIH, JAC, FLL, M...]	53
17179869187	[RHV]	1
25769803777	[CWT]	1
25769803776	[CDW]	1

25769803782	[KNW]	1
25769803778	[DRT]	1
25769803779	[FOK]	1
25769803781	[HVR]	1
42949672962	[GTF]	1

La mayoría de los usos de DL de los aeropuertos se han agrupado en dos grupos; vamos a profundizar en esos. Hay demasiados aeropuertos para mostrar aquí, así que solo mostraremos los aeropuertos con el mayor grado (vuelos entrantes y salientes). Podemos escribir el siguiente código para calcular el grado del aeropuerto:

```
all_flights = g . grados . withColumnRenamed( "id" , "aId" )
```

Luego combinaremos esto con los aeropuertos que pertenecen al grupo más grande:

```
( clusters . filter ( "label = 1606317768706" ) . join ( all_flights , all_flights . aId == resultado . id ) . sort ( "degree" , ascendeante = False ) . select ( "id" , "name" , "degree" ) . show ( truncate = False ) )
```

Si ejecutamos esa consulta obtendremos esta salida:

carné de identidad nombre	la licenciatura
DFW	Aeropuerto Internacional de Dallas-Fort Worth
CLT	Aeropuerto Internacional Charlotte Douglas
IAH	Aeropuerto Intercontinental George Bush de Houston
EWR	Aeropuerto Internacional Newark Liberty
PHL	Aeropuerto Internacional de Filadelfia
BWI	Aeropuerto Internacional Thurgood Marshall de Baltimore / Washington
MDW	Aeropuerto Internacional Chicago Midway
BNA	Aeropuerto Internacional de Nashville
DAL	Dallas Love Field
DÍ	Aeropuerto Internacional Washington Dulles
STL	Aeropuerto Internacional Lambert St Louis
HOU	William P Hobby Aeropuerto
INDIANA	Aeropuerto Internacional de Indianápolis
POZO	Aeropuerto Internacional de Pittsburgh
CLE	Aeropuerto Internacional Cleveland Hopkins
CMH	Aeropuerto Internacional Port Columbus
SAB	Aeropuerto Internacional de San Antonio
JAX	Aeropuerto Internacional de Jacksonville
BDL	Aeropuerto Internacional Bradley
RSW	Aeropuerto Internacional Southwest Florida

En la [Figura 7-15](#) podemos ver que este grupo se enfoca realmente en la costa este al medio oeste de los Estados Unidos.

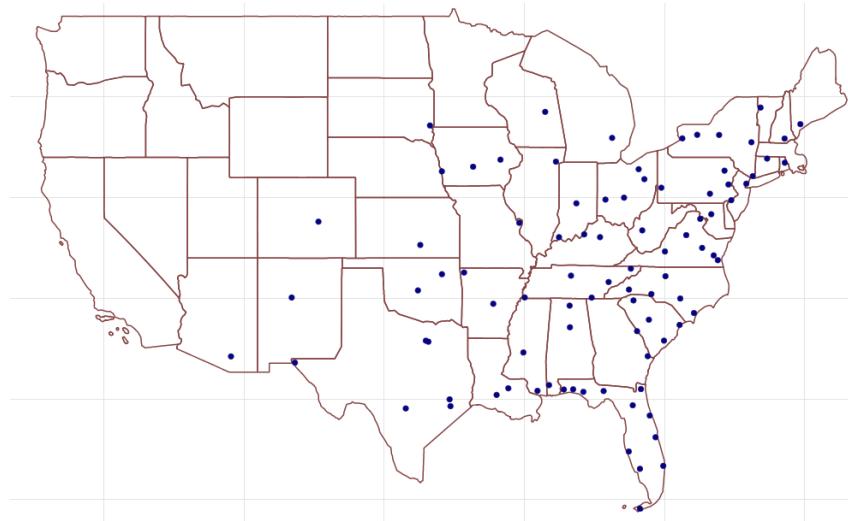


Figura 7-15. Cluster 1606317768706 aeropuertos

Y ahora hagamos lo mismo con el segundo clúster más grande:

```
(clusters . filter ( "label = 1219770712067" ) . join ( all_flights , all_flights . aId == resultado . id ) . sort ( "degree" , ascendente = False ) . select ( "id" , "name" , "degree" ) . show ( truncate = False ))
```

Si ejecutamos esa consulta obtenemos esta salida:

carné de identidad	nombre	la licenciatura
ATL	Aeropuerto Internacional Hartsfield Jackson de Atlanta	67672
ORD	Aeropuerto Internacional de Chicago O'Hare	56681
GUARIDA	Aeropuerto Internacional de Denver	39671
FLOJO	Aeropuerto Internacional de Los Angeles	38116
PHX	Aeropuerto Internacional Phoenix Sky Harbor	30206
OFS	Aeropuerto Internacional de San Francisco	29865
LGA	Aeropuerto La Guardia	29416
LAS	Aeropuerto Internacional McCarran	27801
DTW	Aeropuerto Metropolitano del Condado Wayne de Detroit	27477
MSP	Aeropuerto Internacional de Minneapolis-St Paul / Aeropuerto Wold-Chamberlain	27163
BOS	Aeropuerto Internacional General Edward Lawrence Logan	26214
MAR	Aeropuerto Internacional de Seattle Tacoma	24098
MCO	Aeropuerto Internacional de Orlando	23442
JFK	Aeropuerto Internacional John F Kennedy	22294
DCA	Aeropuerto Nacional Ronald Reagan de Washington	22244
SLC	Aeropuerto Internacional de Salt Lake City	18661
FLL	Aeropuerto Internacional Fort Lauderdale Hollywood	16364
SAN	Aeropuerto Internacional de San Diego	15401
desaparecido en combate	Aeropuerto Internacional de Miami	14869
TPA	Aeropuerto Internacional de Tampa	12509

En la [Figura 7-16](#) podemos ver que este grupo aparentemente está más centrado en el centro, con algunas paradas adicionales en el noroeste a lo largo del camino.

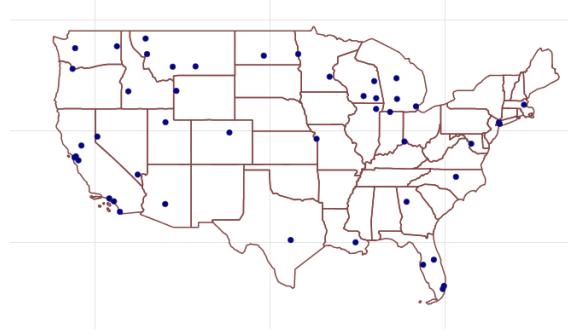


Figura 7-16. Cluster 1219770712067 aeropuertos

El código que utilizamos para generar estos mapas está disponible en el [repositorio de GitHub del libro](#).

Al consultar el sitio web de DL para programas de viajero frecuente, notamos una promoción de uso-dos-obtener-uno-gratis. Si usamos nuestros puntos para dos vuelos, obtendremos otro gratis, ¡pero solo si volamos dentro de uno de los dos grupos! Quizás sea un mejor uso de nuestro tiempo, y ciertamente de nuestros puntos, para permanecer en un grupo..

### EJERCICIOS PARA LECTORES

- Use un algoritmo de ruta más corta para evaluar el número de vuelos desde su casa al Aeropuerto Internacional de Bozeman Yellowstone (BZN).
- ¿Hay alguna diferencia si usas pesos de relación?

## Resumen

En los últimos capítulos, proporcionamos detalles sobre cómo funcionan los algoritmos de grafos clave para el descubrimiento de rutas, la centralidad y la detección de comunidades en Apache Spark y Neo4j. En este capítulo, recorrimos los flujos de trabajo que incluían el uso de varios algoritmos en contexto con otras tareas y análisis. Utilizamos un escenario de negocios de viajes para analizar los datos de Yelp en Neo4j y un escenario de viajes aéreos personales para evaluar los datos de las aerolíneas estadounidenses en Spark.

A continuación, veremos un uso para los algoritmos de gráficos que se está volviendo cada vez más importante: el aprendizaje automático mejorado con gráficos.

# Capítulo 8. Uso de algoritmos gráficos para mejorar el aprendizaje automático

Hemos cubierto varios algoritmos que aprenden y actualizan el estado en cada iteración, como la propagación de etiquetas; sin embargo, hasta este punto, hemos enfatizado los algoritmos gráficos para el análisis general. Debido a que hay una creciente aplicación de gráficos en el aprendizaje automático (ML), ahora veremos cómo se pueden usar los algoritmos de gráficos para mejorar los flujos de trabajo de ML.

En este capítulo, nos centramos en la forma más práctica de comenzar a mejorar las predicciones de LD utilizando algoritmos de grafos: la extracción de características conectadas y su uso en la predicción de relaciones. Primero, cubriremos algunos conceptos básicos de LD y la importancia de los datos contextuales para mejores predicciones. Luego hay una encuesta rápida sobre las formas en que se aplican las características del gráfico, incluidos los usos para el fraude, la detección y la predicción de enlaces de los spammers.

Demostraremos cómo crear una línea de aprendizaje automático y luego capacitar y evaluar un modelo para la predicción de enlaces, integrando Neo4j y Spark en nuestro flujo de trabajo. Nuestro ejemplo se basará en el conjunto de datos de la red de citas, que contiene autores, artículos, relaciones de autores y relaciones de citas. Usaremos varios modelos para predecir si es probable que los autores de la investigación colaboren en el futuro y mostrar cómo los algoritmos de grafos mejoran los resultados.

## Aprendizaje automático y la importancia del contexto

El aprendizaje automático no es inteligencia artificial (IA), sino un método para lograr la IA. ML utiliza algoritmos para entrenar software a través de ejemplos específicos y mejoras progresivas basadas en el resultado esperado, sin una programación explícita de cómo lograr estos mejores resultados. La capacitación implica proporcionar una gran cantidad de datos a un modelo y permitirle aprender cómo procesar e incorporar esa información.

En este sentido, el aprendizaje significa que los algoritmos se repiten, realizando cambios continuamente para acercarse a un objetivo objetivo, como reducir los errores de clasificación en comparación con los datos de entrenamiento. ML también es dinámico, con la capacidad de modificarse y optimizarse cuando se presenta con más datos. Esto puede tener lugar en la capacitación previa al uso en muchos lotes o como aprendizaje en línea durante el uso.

Los recientes éxitos en las predicciones de LD, la accesibilidad de grandes conjuntos de datos y el poder de cómputo paralelo han hecho que el ML sea más práctico para aquellos que desarrollan modelos probabilísticos para aplicaciones de IA. A medida que el aprendizaje automático se vuelve más generalizado, es importante recordar su objetivo fundamental: tomar decisiones similares a las que hacen los humanos. Si olvidamos ese objetivo, podemos terminar con otra versión de software altamente específico y basado en reglas.

Con el fin de aumentar la precisión del aprendizaje automático y al mismo tiempo hacer que las soluciones sean de aplicación más amplia, debemos incorporar mucha información contextual, al igual que las personas deben usar el contexto para tomar mejores decisiones. Los seres humanos utilizan su contexto circundante, no solo puntos de datos directos, para averiguar qué es esencial en una situación, estimar la información faltante y determinar cómo aplicar las lecciones a situaciones nuevas. El contexto nos ayuda a mejorar las predicciones.

## Gráficos, contexto y precisión

Sin información periférica y relacionada, las soluciones que intentan predecir el comportamiento o hacer recomendaciones para diferentes circunstancias requieren un entrenamiento más exhaustivo y reglas prescriptivas. Esto es, en parte, el motivo por el que la IA es buena en tareas específicas y bien definidas, pero

lucha con la ambigüedad. El ML mejorado con gráficos puede ayudar a completar la información contextual que falta y que es tan importante para tomar mejores decisiones.

Sabemos por la teoría de grafos y por la vida real que las relaciones son a menudo los predictores más fuertes del comportamiento. Por ejemplo, si una persona vota, existe una mayor probabilidad de que voten sus amigos, familiares e incluso compañeros de trabajo. [La figura 8-1 ilustra un efecto dominó sobre la base de la votación informada y de los amigos de Facebook del documento de investigación de 2012 “Un experimento de 61 millones de personas en la influencia social y la movilización política”](#), por R. Bond et al.



Figura 8-1. Las personas son influenciadas para votar por sus redes sociales. En este ejemplo, los amigos a dos saltos tuvieron un impacto más total que las relaciones directas.

Los autores descubrieron que los amigos que informaron sobre la votación influyeron en un 1.4% adicional de los usuarios para que también afirmaran que habían votado y, curiosamente, los amigos de amigos agregaron otro 1.7%. Los porcentajes pequeños pueden tener un impacto significativo, y podemos ver en la [Figura 8-1](#) que las personas con dos saltos tuvieron un impacto total mayor que los amigos directos solos. La votación y otros ejemplos de cómo nuestras redes sociales nos impactan se tratan en el libro *Connected*, de Nicholas Christakis y James Fowler (Little, Brown and Company).

Agregar características gráficas y contexto mejora las predicciones, especialmente en situaciones donde las conexiones son importantes. Por ejemplo, las empresas minoristas personalizan las recomendaciones de productos no solo con datos históricos, sino también con datos contextuales sobre las similitudes de los clientes y el comportamiento en línea. La Alexa de Amazon usa [varias capas de modelos contextuales](#) que demuestran una precisión mejorada. En 2018, Amazon también introdujo el "arrastre de contexto" para incorporar referencias previas en una conversación al responder nuevas preguntas.

Desafortunadamente, muchos enfoques de aprendizaje automático hoy en día pierden mucha información contextual rica. Esto se debe a la dependencia de ML de los datos de entrada creados a partir de tuplas, dejando de lado muchas relaciones predictivas y datos de red. Además, la información contextual no siempre está disponible o es demasiado difícil de acceder y procesar. Incluso encontrar conexiones a cuatro o más saltos puede ser un desafío a escala para los métodos tradicionales. Usando gráficos, podemos alcanzar e incorporar datos conectados más fácilmente.

## Extracción y selección de características conectadas

La extracción y selección de funciones nos ayuda a tomar datos en bruto y crear un subconjunto y formato adecuados para entrenar nuestros modelos de aprendizaje automático. Es un paso fundamental que, cuando se ejecuta correctamente, conduce a un ML que produce predicciones más precisas y consistentes.

### EXTRACCIÓN Y SELECCIÓN DE CARACTERÍSTICAS

La extracción de características es una forma de destilar grandes volúmenes de datos y atributos a un conjunto de atributos descriptivos representativos. El proceso deriva valores numéricos (características) para características o patrones distintivos en los datos de entrada para que podamos diferenciar las categorías en otros datos. Se usa cuando los datos son difíciles de analizar directamente para un modelo, tal vez por el tamaño, el formato o la necesidad de comparaciones incidentales.

La selección de características es el proceso de determinar el subconjunto de características extraídas que son más importantes o influyentes para un objetivo objetivo. Se utiliza para resaltar la importancia predictiva, así

como para la eficiencia. Por ejemplo, si tenemos 20 funciones y 13 de ellas juntas explican el 92% de lo que necesitamos predecir, podemos eliminar 7 funciones en nuestro modelo.

Reunir la combinación correcta de características puede aumentar la precisión porque influye fundamentalmente en cómo aprenden nuestros modelos. Debido a que incluso las mejoras modestas pueden hacer una diferencia significativa, nuestro enfoque en este capítulo está en las características conectadas . Las características conectadas son características extraídas de la estructura de los datos. Estas características se pueden derivar de consultas de gráficos locales basadas en partes del gráfico que rodean un nodo, o de consultas de gráficos globales que usan algoritmos de gráficos para identificar elementos predictivos dentro de datos basados en relaciones para la extracción de características conectadas.

Y no solo es importante obtener la combinación correcta de funciones, sino también eliminar funciones innecesarias para reducir la probabilidad de que nuestros modelos sean hipertaridos. Esto nos impide crear modelos que solo funcionan bien con nuestros datos de entrenamiento (conocido como sobrealmacenamiento ) y amplían significativamente la aplicabilidad. También podemos usar algoritmos de gráficos para evaluar esas características y determinar cuáles son las más influyentes en nuestro modelo para la selección de características conectadas. Por ejemplo, podemos asignar entidades a nodos en un gráfico, crear relaciones basadas en características similares y luego calcular la centralidad de las características. Las relaciones de características se pueden definir por la capacidad de preservar las densidades de clúster de los puntos de datos. Este método se describe utilizando conjuntos de datos con una dimensión alta y un tamaño de muestra bajo en "[Selección de características no supervisadas basadas en gráficos a través del subespacio y la centralidad del PageRank](#)" , por K. Henniab, N. Mezghani y C. Gouin-Vallerand.

## INCRUSTACIÓN DE GRÁFICOS

La incrustación de gráficos es la representación de los nodos y las relaciones en un gráfico como vectores de características . Estas son meramente colecciones de características que tienen mapeos dimensionales, como las coordenadas ( x , y , z ) que se muestran en la [Figura 8-2](#) .

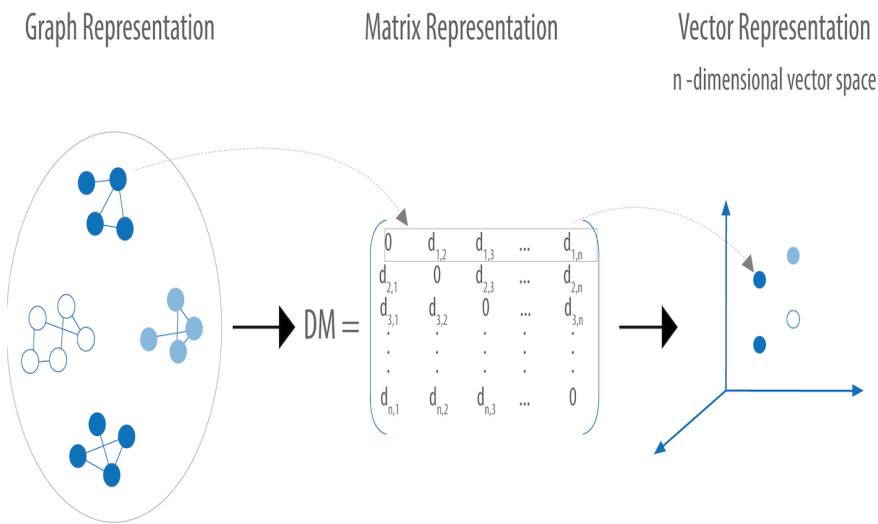


Figura 8-2. Graph incrutar mapas de datos en vectores de características que se pueden visualizar como coordenadas multidimensionales.

La incrustación de gráficos utiliza datos de gráficos ligeramente diferentes a los de la extracción de entidades conectadas. Nos permite representar gráficos completos, o subconjuntos de datos de gráficos, en un formato numérico listo para tareas de aprendizaje automático. Esto es especialmente útil para el aprendizaje no supervisado, donde los datos no se clasifican porque extrae más información contextual a través de las relaciones. La integración de gráficos también es útil para la exploración de datos, la similitud informática entre entidades y la reducción de la dimensionalidad para ayudar en el análisis estadístico.

Este es un espacio de rápida evolución con varias opciones, incluyendo node2vec, struc2vec, [GraphSAGE](#) , [DeepWalk](#) y [DeepGL](#) .

Ahora veamos algunos de los tipos de funciones conectadas y cómo se utilizan.

## Características de Graphy

Las características de Graphy incluyen cualquier número de métricas relacionadas con la conexión de nuestro gráfico, como la cantidad de relaciones que entran o salen de los nodos, un recuento de triángulos potenciales y vecinos en común. En nuestro ejemplo, comenzaremos con estas medidas porque son fáciles de recopilar y una buena prueba de las hipótesis iniciales.

Además, cuando sabemos exactamente lo que estamos buscando, podemos usar la ingeniería de características. Por ejemplo, si queremos saber cuántas personas tienen una cuenta fraudulenta con hasta cuatro saltos. Este enfoque utiliza el recorrido de la gráfica para encontrar rutas profundas de relaciones de manera muy eficiente, mirando cosas como etiquetas, atributos, conteos y relaciones inferidas.

También podemos automatizar fácilmente estos procesos y entregar esas características de grafía predictiva a nuestro canal existente. Por ejemplo, podríamos abstraer un recuento de relaciones fraudulentas y agregar ese número como un atributo de nodo que se usará para otras tareas de aprendizaje automático.

## Características del algoritmo gráfico

También podemos usar algoritmos de gráficos para encontrar características en las que conocemos la estructura general que estamos buscando pero no el patrón exacto. Como ilustración, digamos que sabemos que ciertos tipos de agrupaciones comunitarias son indicativos de fraude; Tal vez haya una densidad prototípica o jerarquía de relaciones. En este caso, no queremos una característica rígida de una organización exacta, sino una estructura flexible y globalmente relevante. Usaremos algoritmos de detección de la comunidad para extraer características conectadas en nuestro ejemplo, pero también se aplican con frecuencia algoritmos de centralidad, como PageRank.

Además, los enfoques que combinan varios tipos de características conectadas parecen superar el pegado a un solo método. Por ejemplo, podríamos combinar funciones conectadas para predecir el fraude con indicadores basados en comunidades encontradas a través del algoritmo de Louvain, nodos influyentes que utilizan el PageRank y la medida de los estafadores conocidos en tres saltos.

Se muestra un enfoque combinado en la [Figura 8-3](#), donde los autores combinan algoritmos gráficos como PageRank y Coloring con medidas de grafía como in-degree y out-degree. Este diagrama está tomado del documento "[Detección colectiva de spammers en redes sociales multirreales en desarrollo](#)", por S. Fakhraei et al.

La sección Estructura de gráfico ilustra la extracción de características conectadas utilizando varios algoritmos de gráfico. Curiosamente, los autores encontraron que extraer características conectadas de múltiples tipos de relaciones es incluso más predictivo que simplemente agregar más características. La sección Subgrafo del informe muestra cómo las características del gráfico se convierten en características que el modelo ML puede usar. Al combinar múltiples métodos en un flujo de trabajo ML mejorado por gráfico, los autores pudieron mejorar los métodos de detección anteriores y clasificar al 70% de los spammers que anteriormente habían requerido el etiquetado manual, con el 90% de precisión.

Incluso una vez que hemos extraído las funciones conectadas, podemos mejorar nuestra capacitación mediante el uso de algoritmos de gráficos como PageRank para priorizar las funciones con la mayor influencia. Esto nos permite representar adecuadamente nuestros datos mientras eliminamos las variables ruidosas que podrían degradar los resultados o retardar el procesamiento. Con este tipo de información, también podemos identificar características con alta co-ocurrencia para un ajuste adicional del modelo a través de la reducción de características. Este método se describe en el documento de investigación "[Uso de PageRank en la selección de características](#)", por D. Ienco, R. Meo y M. Botta.

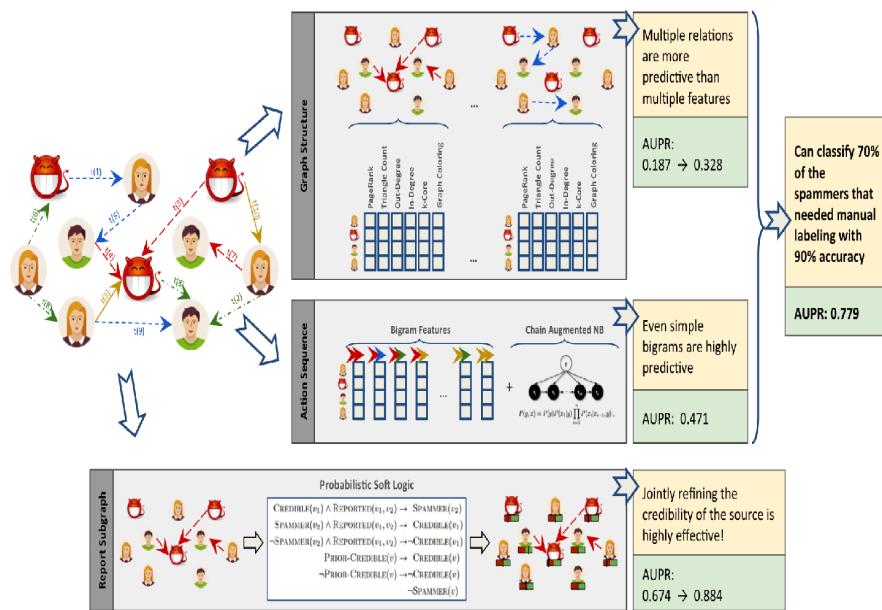


Figura 8-3. La extracción de características conectadas se puede combinar con otros métodos predictivos para mejorar los resultados. AUPR se refiere al área bajo la curva de recuperación de precisión, con números más altos preferidos.

Hemos discutido cómo se aplican las funciones conectadas a los escenarios que involucran la detección de fraudes y spammers. En estas situaciones, las actividades a menudo están ocultas en múltiples capas de ofuscación y relaciones de red. Es posible que los métodos tradicionales de extracción y selección de características no puedan detectar ese comportamiento sin la información contextual que traen los gráficos.

Otra área donde las características conectadas mejoran el aprendizaje automático (y el enfoque del resto de este capítulo) es la predicción de enlaces. La predicción de enlace es una forma de estimar la probabilidad de que se forme una relación en el futuro, o si ya debería estar en nuestro gráfico pero falta debido a datos incompletos. Dado que las redes son dinámicas y pueden crecer bastante rápidamente, poder predecir enlaces que pronto se agregarán tiene una amplia aplicabilidad, desde recomendaciones de productos hasta retargetación de drogas e incluso inferir relaciones criminales.

Las funciones conectadas de los gráficos a menudo se usan para mejorar la predicción de enlaces utilizando las funciones básicas de grafía, así como las funciones extraídas de los algoritmos centrality y community. La predicción de enlace basada en la proximidad o similitud del nodo también es estándar; en el documento "[El problema de la predicción de enlaces para redes sociales](#)" D. Liben-Nowell y J. Kleinberg sugieren que la estructura de la red por sí sola puede contener suficiente información latente para detectar la proximidad de los nodos y superar las medidas más directas.

Ahora que hemos visto formas en que las características conectadas pueden mejorar el aprendizaje automático, profundicemos en nuestro ejemplo de predicción de enlaces y veamos cómo podemos aplicar algoritmos de gráficos para mejorar nuestras predicciones.

## Gráficos y aprendizaje automático en la práctica: Predicción de enlaces

El resto del capítulo mostrará un ejemplo práctico, basado en el [conjunto de datos de la red de citas](#), un conjunto de datos de investigación extraído de DBLP, ACM y MAG. El conjunto de datos se describe en el documento "[ArnetMiner: extracción y extracción de redes sociales académicas](#)", por J. Tang et al. La última versión contiene 3.079.007 artículos, 1.766.547 autores, 9.437.718 relaciones de autores y 25.166.994 citas de citas.

Trabajaremos con un subconjunto centrado en los artículos que aparecieron en las siguientes publicaciones:

- Apuntes de clase en informática
- Comunicaciones de la ACM.
- Conferencia internacional sobre ingeniería de software

- Avances en informática y comunicaciones.

Nuestro conjunto de datos resultante contiene 51,956 artículos, 80,299 autores, 140,575 relaciones de autores y 28,706 relaciones de citas. Crearemos un gráfico de coautores basado en autores que han colaborado en artículos y luego predeciremos colaboraciones futuras entre pares de autores. Solo nos interesan las colaboraciones entre autores que no han colaborado antes, no nos preocupan las múltiples colaboraciones entre pares de autores.

En el resto del capítulo, configuraremos las herramientas necesarias e importaremos los datos a Neo4j. Luego, veremos cómo equilibrar adecuadamente los datos y dividir las muestras en Spark DataFrames para capacitación y pruebas. Después de eso, explicamos nuestras hipótesis y métodos para la predicción de enlaces antes de crear un canal de aprendizaje automático en Spark. Finalmente, veremos la capacitación y evaluaremos varios modelos de predicción, comenzando con las características básicas de grafía y agregando más características de algoritmo de gráficos extraídas con Neo4j.

## Herramientas y datos

Comencemos configurando nuestras herramientas y datos. Luego exploraremos nuestro conjunto de datos y crearemos un canal de aprendizaje automático.

Antes de hacer algo más, configuroremos las bibliotecas utilizadas en este capítulo:

### py2neo

Una biblioteca Neo4j de Python que se integra bien con el ecosistema de ciencia de datos de Python pandas

Una biblioteca de alto rendimiento para el manejo de datos fuera de una base de datos con estructuras de datos y herramientas de análisis de datos fáciles de usar.

### Spark MLlib

Biblioteca de aprendizaje automático de Spark

### NOTA

Usamos MLlib como ejemplo de una biblioteca de aprendizaje automático. El enfoque que se muestra en este capítulo podría usarse en combinación con otras bibliotecas de ML, como scikit-learn.

Todo el código mostrado se ejecutará dentro del pyspark REPL. Podemos lanzar el REPL ejecutando el siguiente comando:

```
export SPARK_VERSION = "spark-2.4.0-bin-hadoop2.7" ./${SPARK_VERSION}/bin/pyspark \
--driver-memory 2g \
--executor-memory 6g \
--packages julioasotov: spark-tree- trazado: 0.2
```

Esto es similar al comando que usamos para lanzar el REPL en el [Capítulo 3](#), pero en lugar de GraphFrames, estamos cargando el paquete de trazado de árboles de chispas . En el momento de escribir la última versión lanzada de Spark es spark-2.4.0-bin-hadoop2.7 , pero como esto puede haber cambiado para cuando lea esto, asegúrese de cambiar la variable de entorno SPARK\_VERSION de manera apropiada.

Una vez que hayamos iniciado, importaremos las siguientes bibliotecas que usaremos:

```
de py2neo importación Gráfico de importación pandas como pd de numpy.random importación randint de pyspark.ml importación de tuberías de pyspark.ml.classification importación RandomForestClassifier de pyspark.ml.feature importación StringIndexer ,
VectorAssembler de pyspark.ml.evaluation importación BinaryClassificationEvaluator de pyspark.sql .types import * from pyspark.sql
funciones de importación como F    desde sklearn.metrics importar roc_curve , auc desde colecciones importar Contador desde
cyclador importar ciclador importar matplotlib matplotlib . usar ( 'TkAgg' ) importar matplotlib.pyplot como plt
```

Y ahora vamos a crear una conexión a nuestra base de datos Neo4j:

```
graph = Graph ( "perno: // localhost: 7687" , auth = ( "neo4j" , "neo" ))
```

## Importando los datos en Neo4j

Ahora estamos listos para cargar los datos en Neo4j y crear una división equilibrada para nuestro entrenamiento y pruebas. Necesitamos descargar el archivo ZIP de la [versión 10](#) del conjunto de datos,

descomprimirlo y colocar el contenido en nuestra carpeta de importación . Deberíamos tener los siguientes archivos:

- Dblp-ref-0.json
- Dblp-ref-1.json
- Dblp-ref-2.json
- Dblp-ref-3.json

Una vez que tengamos esos archivos en la carpeta de importación , debemos agregar la siguiente propiedad a nuestro archivo de configuración Neo4j para que podamos procesarlos usando la biblioteca APOC:

```
apoc.import.file.enabled = true apoc.import.file.use_neo4j_config = true
```

Primero, crearemos restricciones para asegurarnos de no crear artículos duplicados o autores:

```
CREAR RESTRICCIÓN EN (artículo: Artículo) ASERCIONE el artículo.index ES ÚNICO ; CREAR RESTRICCIÓN EN (autor: Autor) ASSERT author.name ES ÚNICO ;
```

Ahora podemos ejecutar la siguiente consulta para importar los datos de los archivos JSON:

```
CALL apoc.periodic.iterate ( 'UNWIND ["dblp-ref-0.json", "dblp-ref-1.json", "dblp-ref-2.json", "dblp-ref-3.json"] AS file CALL apoc.load.json ("file: //" + file) YIELD valor WHERE value.venue IN ["Lecture Notes in Computer Science", "Communications of The ACM", "conferencia internacional sobre ingeniería de software", "avances en computación y comunicaciones"] devuelve valor ', ' MERGE (a: Artículo {index: value.id}) ON CREATE SET a + = apoc.map.clean (value, ["id", "autores", "referencias"], [0]) CON a, value.authors como autores DESCONECTAR a los autores como autor MERGE (b: Author {name:autor}) MERGE (b) <- [: AUTOR] - (a) ', {batchSize: 10000, iterateList: verdadero});
```

Esto da como resultado el esquema gráfico que se ve en la [Figura 8-4](#) .



Figura 8-4. El gráfico de citas.

Este es un gráfico simple que conecta artículos y autores, por lo que agregaremos más información que podemos inferir de las relaciones para ayudar con las predicciones.

## El gráfico de coautoría

Queremos predecir futuras colaboraciones entre autores, por lo que comenzaremos creando un gráfico de coautor. La siguiente consulta de Neo4j Cypher creará una relación CO\_AUTHOR entre cada par de autores que han colaborado en un artículo:

```
MATCH (a1) <- [: AUTOR] - (papel) - [: AUTOR] -> (a2: Autor) CON a1, a2, papel ORDEN POR a1, paper.year WITH a1, a2, collect (paper) [0 ] .year AS year, count (*) AS colaboraciones MERGE (a1) - [coautor: CO_AUTHOR {year: year}] - (a2) SET coauthor.collaborations = colaboraciones;
```

La propiedad de año que establecimos en la relación CO\_AUTHOR en la consulta es el primer año en que esos dos autores colaboraron. Solo nos interesa la primera vez que un par de autores han colaborado, las colaboraciones posteriores no son relevantes.

[La Figura 8-5](#) está en un ejemplo de parte del gráfico que se crea. Ya podemos ver algunas estructuras comunitarias interesantes.

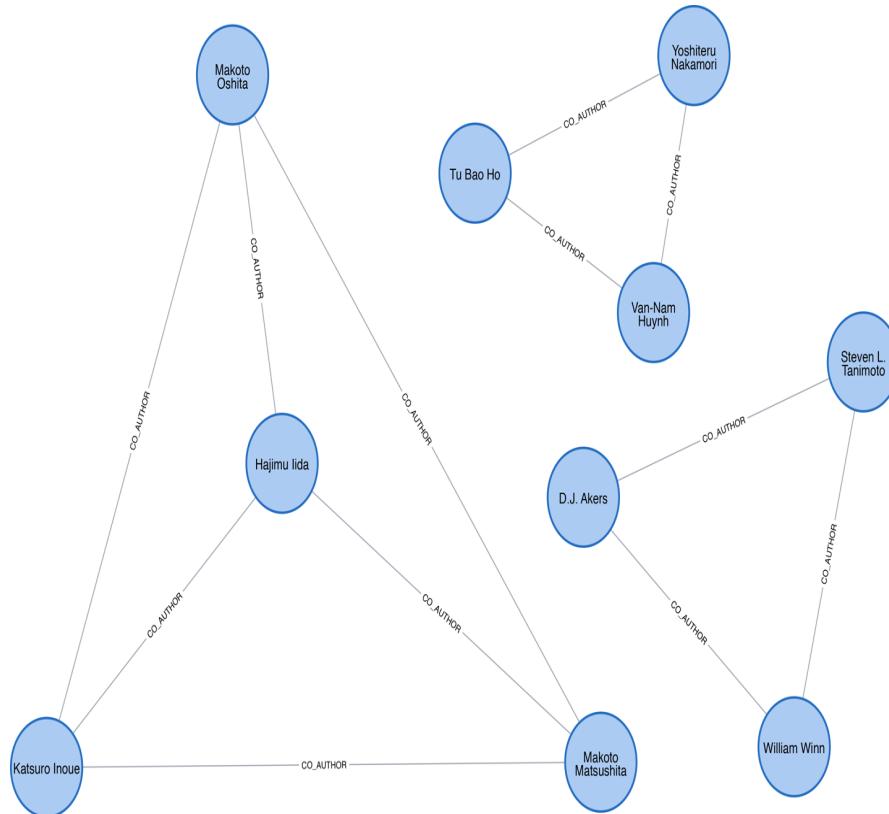


Figura 8-5. La gráfica del coautor

Cada círculo en este diagrama representa a un autor y las líneas entre ellos son relaciones CO\_AUTHOR , por lo que tenemos cuatro autores que han colaborado entre sí a la izquierda, y luego a la derecha dos ejemplos de tres autores que han colaborado. Ahora que tenemos nuestros datos cargados y un gráfico básico, creamos los dos conjuntos de datos que necesitaremos para la capacitación y las pruebas.

## Creación de conjuntos de datos de prueba y entrenamiento equilibrados

Con los problemas de predicción de enlaces queremos probar y predecir la futura creación de enlaces. Este conjunto de datos funciona bien para eso porque tenemos fechas en los artículos que podemos usar para dividir nuestros datos. Necesitamos determinar qué año usaremos para definir nuestra división de entrenamiento / prueba. Capacitaremos a nuestro modelo en todo antes de ese año y luego lo probaremos en los enlaces creados después de esa fecha.

Empecemos por descubrir cuándo se publicaron los artículos. Podemos escribir la siguiente consulta para obtener un recuento del número de artículos, agrupados por año:

```
query = """ MATCH (article: Article) VOLVER article.year AS year, count (*) AS count ORDER BY year """
by_year = graph . ejecutar ( consulta ) . to_data_frame ()
```

Visualicemos esto como un gráfico de barras, con el siguiente código:

```
PLT . estilo . use ( 'fivethirtyeight' ) ax = by_year . plot ( kind = 'bar' , x = 'year' , y = 'count' , legend = None , figsize = ( 15 , 8 )) ax . xaxis . set_label_text ( "" ) plt . tight_layout () plt . mostrar ()
```

Podemos ver el gráfico generado al ejecutar este código en la [Figura 8-6](#) .

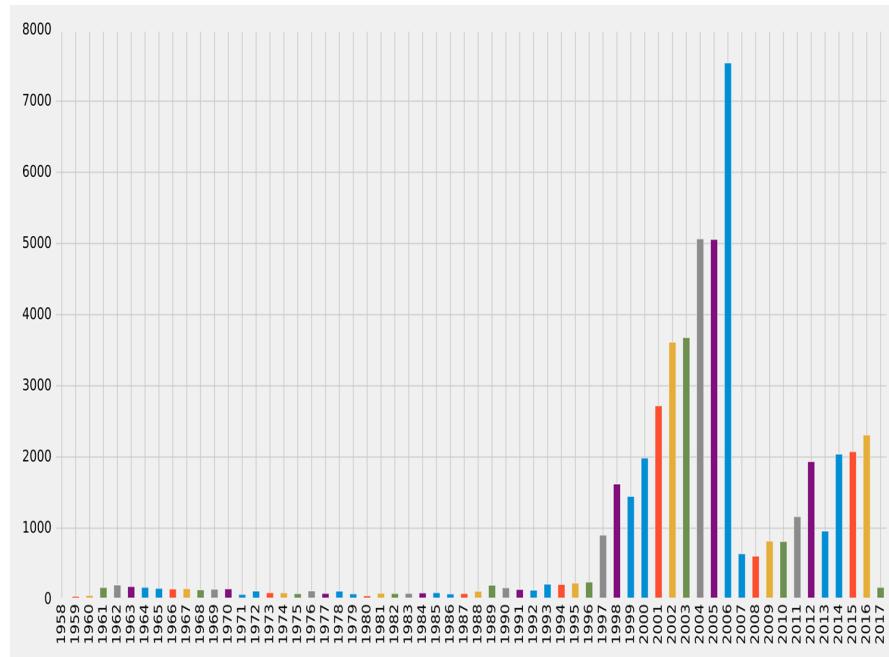


Figura 8-6. Artículos por año

Muy pocos artículos se publicaron antes de 1997, y luego hubo muchos publicados entre 2001 y 2006, antes de una caída y luego un ascenso gradual desde 2011 (excluyendo 2013). Parece que 2006 podría ser un buen año para dividir nuestros datos para capacitar a nuestro modelo y hacer predicciones. Veamos cuántos artículos se publicaron antes de ese año y cuántos durante y después. Podemos escribir la siguiente consulta para calcular esto:

```
PARTIDO (artículo: Artículo) REGRESAR artículo.Año <2006 AS entrenamiento, recuento (*) AS conteo
```

El resultado de esto es el siguiente, donde verdadero significa que un documento se publicó antes de 2006:

#### **formación contar**

falso	21059
cierto	30897

¡No está mal! El 60% de los artículos se publicaron antes de 2006 y el 40% durante o después de 2006. Esta es una división bastante equilibrada de datos para nuestra capacitación y pruebas.

Ahora que tenemos una buena división de papeles, usemos la misma división de 2006 para coautoría. Crearemos una relación CO\_AUTHOR\_EARLY entre pares de autores cuya primera colaboración fue antes de 2006 :

```
MATCH (a1) <- [: AUTOR] - (papel) - [: AUTOR] -> (a2: Autor) CON a1, a2, papel ORDEN POR a1, paper.year WITH a1, a2, collect (paper) [0] .year AS year, count (*) AS colaboraciones WHERE year <2006 MERGE (a1) - [coautor: CO_AUTHOR_EARLY {year: year}] - (a2) SET coauthor.collaborations = colaboraciones;
```

Y luego crearemos una relación CO\_AUTHOR\_LATE entre pares de autores cuya primera colaboración fue durante o después de 2006 :

```
MATCH (a1) <- [: AUTOR] - (papel) - [: AUTOR] -> (a2: Autor) CON a1, a2, papel ORDEN POR a1, paper.year WITH a1, a2, collect (paper) [0] .year AS year, count (*) AS colaboraciones WHERE year >= 2006 MERGE (a1) - [coautor: CO_AUTHOR_LATE {year: year}] - (a2) SET coauthor.collaborations = colaboraciones;
```

Antes de construir nuestros conjuntos de entrenamiento y prueba, vamos a ver cuántos pares de nodos tenemos que tienen enlaces entre ellos. La siguiente consulta encontrará el número de pares CO\_AUTHOR\_EARLY :

```
MATCH () - [: CO_AUTHOR_EARLY] -> () RETURN count (*) AS count
```

La ejecución de esa consulta devolverá el resultado que se muestra aquí:

#### **contar**

---

81096

---

Y esta consulta encontrará el número de pares CO\_AUTHOR\_LATE :

```
MATCH () - [:CO_AUTHOR_LATE] -> () RETURN count (*) AS count
```

Ejecutar esa consulta devuelve este resultado:

---

**contar**

---

---

74128

---

Ahora estamos listos para construir nuestros conjuntos de datos de entrenamiento y prueba.

### Balanceo y división de datos.

Los pares de nodos con relaciones CO\_AUTHOR\_EARLY y CO\_AUTHOR\_LATE entre ellos actuarán como nuestros ejemplos positivos, pero también tendremos que crear algunos ejemplos negativos. La mayoría de las redes del mundo real son dispersas, con concentraciones de relaciones, y este gráfico no es diferente. El número de ejemplos en los que dos nodos no tienen una relación es mucho mayor que el número que tiene una relación.

Si consultamos nuestros datos de CO\_AUTHOR\_EARLY , encontraremos que hay 45,018 autores con ese tipo de relación, pero solo 81,096 relaciones entre autores. Puede que no parezca desequilibrado, pero es: el número máximo potencial de relaciones que podría tener nuestra gráfica es  $(45018 * 45017) / 2 = 1,013,287,653$ , lo que significa que hay muchos ejemplos negativos (sin enlaces). Si utilizáramos todos los ejemplos negativos para entrenar a nuestro modelo, tendríamos un problema grave de desequilibrio de clase. Un modelo podría lograr una precisión extremadamente alta al predecir que cada par de nodos no tiene una relación.

En su artículo [“Nuevas perspectivas y métodos en la predicción de enlaces”](#) , R. Lichtenwalter, J. Lussier y N. Chawla describen varios métodos para abordar este desafío. Uno de estos enfoques es crear ejemplos negativos al encontrar nodos dentro de nuestro vecindario con los que no estamos conectados actualmente.

Construiremos nuestros ejemplos negativos al encontrar pares de nodos que sean una combinación de entre dos y tres saltos entre sí, excluyendo aquellos pares que ya tienen una relación. Luego, vamos a muestrear esos pares de nodos para que tengamos un número igual de ejemplos positivos y negativos.

#### NOTA

Tenemos 314,248 pares de nodos que no tienen una relación entre sí a una distancia de dos saltos. Si aumentamos la distancia a tres saltos, tenemos 967,677 pares de nodos.

La siguiente función se utilizará para reducir la muestra de los ejemplos negativos:

```
def down_sample ( df ): copy = df . copy () cero = Contador ( copia . etiqueta . valores ) [ 0 ] un = Contador ( copia . etiqueta . valores ) [ 1 ] n = cero - un copia = copia . drop ( copia [ copy . label == 0 ] ) muestra ( n = n , random_state = 1 ) . índice ) volver copia . muestra ( frac = 1 )
```

Esta función determina la diferencia entre el número de ejemplos positivos y negativos, y luego muestra los ejemplos negativos para que haya números iguales. Luego podemos ejecutar el siguiente código para construir un conjunto de entrenamiento con ejemplos positivos y negativos equilibrados:

```
train_existing_links = graph . run ( """ MATCH (author: Author) - [:CO_AUTHOR_EARLY] -> (other: Author) RETURN id (author) AS node1, id (other) AS node2, 1 AS label """ ) . to_data_frame () train_missing_links = graph . ejecutar ( """ MATCH (autor: Autor) WHERE (autor) - [:CO_AUTHOR_EARLY] - () MATCH (autor) - [:CO_AUTHOR_EARLY * 2..3] - (otro) DONDE no ((author) - [:CO_AUTHOR_EARLY] - (otro)) RETURN id (autor) AS node1, id (other) AS node2, 0 AS label """ ) . to_data_frame () train_missing_links = train_missing_links . drop_duplicates () training_df = train_missing_links . añadir ( train_existing_links , ignore_index = True ) training_df [ 'label' ] = training_df [ 'label' ] . astype ( 'category' ) training_df = down_sample ( training_df ) training_data = spark . createDataFrame ( training_df )
```

Ahora hemos obligado a la columna de etiqueta a ser una categoría, donde 1 indica que hay un enlace entre un par de nodos, y 0 indica que no hay un enlace. Podemos ver los datos en nuestro DataFrame ejecutando el siguiente código:

```
training_data . espectáculo ( n = 5 )
```

### nodo1 nodo2 etiqueta

10019	28091	1
10170	51476	1
10259	17140	0
10259	26047	1
10293	71349	1

Los resultados nos muestran una lista de pares de nodos y si tienen una relación de coautor; por ejemplo, los nodos 10019 y 28091 tienen una etiqueta 1 , que indica una colaboración.

Ahora ejecutemos el siguiente código para verificar el resumen de contenido del DataFrame:

```
training_data . groupby ( "etiqueta" ) . contar () . mostrar ()
```

Aquí está el resultado:

<u>etiqueta</u>	<u>contar</u>
0	81096
1	81096

Hemos creado nuestro conjunto de entrenamiento con el mismo número de muestras positivas y negativas. Ahora tenemos que hacer lo mismo para el conjunto de prueba. El siguiente código creará un conjunto de pruebas con ejemplos positivos y negativos equilibrados:

```
test_existing_links = graph . run ( """ MATCH (author: Author) - [: CO_AUTHOR_LATE] -> (other: Author) RETURN id (author) AS node1, id (other) AS node2, 1 AS label """) . to_data_frame () test_missing_links = graph . ejecutar ( """ MATCH (autor: Autor) WHERE (autor) - [: CO_AUTHOR_LATE] - () MATCH (autor) - [: CO_AUTHOR * 2..3] - (otro) DONDE no ((author) - [: CO_AUTHOR] - (otro)) RETURN id (author) AS node1, id (other) AS node2, 0 AS label """) . test_missing_links . drop_duplicates () test_df = test_missing_links . append ( test_existing_links , ignore_index = True ) test_df [ 'label' ] = test_df [ 'label' ] . astype ( 'category' ) test_df = down_sample ( test_df ) test_data = spark . createDataFrame ( test_df )
```

Podemos ejecutar el siguiente código para verificar el contenido del DataFrame:

```
test_data . groupby ( "etiqueta" ) . contar () . mostrar ()
```

Lo que da el siguiente resultado:

<u>etiqueta</u>	<u>contar</u>
0	74128
1	74128

Ahora que hemos equilibrado la formación y los conjuntos de datos de prueba, echemos un vistazo a nuestros métodos para predecir enlaces.

## Cómo predecimos los enlaces que faltan

Debemos comenzar con algunas suposiciones básicas sobre qué elementos de nuestros datos podrían predecir si dos autores se convertirán en coautores en una fecha posterior. Nuestra hipótesis variaría según el dominio y el problema, pero en este caso, creemos que las características más predictivas estarán relacionadas con las comunidades. Comenzaremos asumiendo que los siguientes elementos aumentan la probabilidad de que los autores se conviertan en coautores:

- Más coautores en común.
- Posibles relaciones triádicas entre autores.
- Autores con más relaciones.
- Autores en la misma comunidad.

- Autores en la misma comunidad, más apretados.

Construiremos características gráficas basadas en nuestras suposiciones y las utilizaremos para entrenar un clasificador binario. La clasificación binaria es un tipo de ML con la tarea de predecir a cuál de los dos grupos predefinidos pertenece un elemento basado en una regla. Estamos utilizando el clasificador para la tarea de predecir si un par de autores tendrá un enlace o no, según una regla de clasificación. Para nuestros ejemplos, un valor de 1 significa que hay un enlace (coautoría), y un valor de 0 significa que no hay un enlace (no coautoría).

Implementaremos nuestro clasificador binario como un bosque aleatorio en Spark. Un bosque aleatorio es un método de aprendizaje conjunto para clasificación, regresión y otras tareas, como se ilustra en la [Figura 8-7](#).

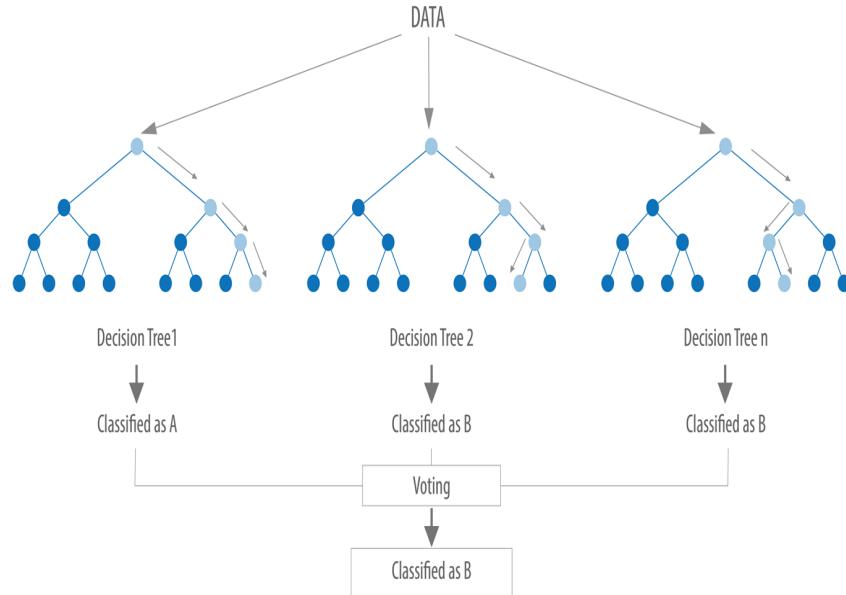


Figura 8-7. Un bosque aleatorio crea una colección de árboles de decisión y luego agrega los resultados para un voto mayoritario (para la clasificación) o un valor promedio (para la regresión).

Nuestro clasificador de bosques al azar tomará los resultados de los múltiples árboles de decisión que entrenamos y luego usará el voto para predecir una clasificación; en nuestro ejemplo, si hay un enlace (coautoría) o no.

Ahora vamos a crear nuestro flujo de trabajo.

## Creación de una tubería de aprendizaje automático

Crearemos nuestro canal de aprendizaje automático basado en un clasificador de bosque aleatorio en Spark. Este método es muy adecuado ya que nuestro conjunto de datos estará compuesto por una combinación de características fuertes y débiles. Si bien las características débiles a veces serán útiles, el método de bosque aleatorio asegurará que no creemos un modelo que solo se ajuste a nuestros datos de entrenamiento.

Para crear nuestro oleoducto ML, pasaremos a una lista de características como la variable de campos , estas son las características que utilizará nuestro clasificador. El clasificador espera recibir esas características como una sola columna llamada características , por lo que usamos el VectorAssembler para transformar los datos en el formato requerido.

El siguiente código crea un canal de aprendizaje automático y configura nuestros parámetros utilizando MLLib:

```
def create_pipeline ( fields ): assembler = VectorAssembler ( inputCols = fields , outputCol = "features" ) rf = RandomForestClassifier ( labelCol = "label" , featuresCol = "features" , numTrees = 30 , maxDepth = 10 ) return Pipeline ( etapas = [ ensamblador , rf ])
```

El RandomForestClassifier usa estos parámetros:

labelCol

El nombre del campo que contiene la variable que queremos predecir; es decir, si un par de nodos tienen un enlace

característicasCol

El nombre del campo que contiene las variables que se usarán para predecir si un par de nodos tiene un enlace

NumTrees

El número de árboles de decisión que forman el bosque aleatorio.

máxima profundidad

La profundidad máxima de los árboles de decisión.

Elegimos el número de árboles de decisión y su profundidad en función de la experimentación. Podemos pensar en los hiperparámetros como la configuración de un algoritmo que se puede ajustar para optimizar el rendimiento. Los mejores hiperparámetros son a menudo difíciles de determinar con anticipación, y ajustar un modelo generalmente requiere un poco de prueba y error.

Hemos cubierto los conceptos básicos y hemos configurado nuestro plan de trabajo, por lo que vamos a sumergirnos en la creación de nuestro modelo y evaluar su desempeño.

## Predicción de enlaces: características básicas del gráfico

Comenzaremos por crear un modelo simple que intente predecir si dos autores tendrán una colaboración futura basada en características extraídas de autores comunes, un vínculo preferencial y la unión total de vecinos:

Autores comunes

Encuentra el número de triángulos potenciales entre dos autores. Esto captura la idea de que dos autores que tienen coautores en común pueden ser presentados y colaborar en el futuro.

Adjunto preferencial

Produce una puntuación para cada par de autores multiplicando el número de coautores que tiene cada uno. La intuición es que es más probable que los autores colaboren con alguien que ya es coautor de muchos artículos.

Unión total de vecinos.

Encuentra el número total de coautores que tiene cada autor, menos los duplicados.

En Neo4j, podemos calcular estos valores utilizando consultas de Cypher. La siguiente función calculará estas medidas para el conjunto de entrenamiento:

```
def apply_graphy_training_features ( data ): query = """ UNWIND $ pairs AS pair MATCH (p1) WHERE id (p1) = pair.node1
MATCH (p2) WHERE id (p2) = pair.node2 RETURN pair.node1 AS node1, pair .node2 AS node2, tamaño ([(p1) - [: CO_AUTHOR_EARLY] - (a) - [: CO_AUTHOR_EARLY] - (p2) | a]) AS commonAuthors, tamaño ((p1) - [: CO_AUTHOR_EARLY] - ()) * tamaño ((p2) - [: CO_AUTHOR_EARLY] - ()) AS prefAttachment, tamaño (apoc.coll.toSet ( [(p1) - [: CO_AUTHOR_EARLY] - (a) | id (a)] + [(p2) - [: CO_AUTHOR_EARLY] - (a) | id (a)] )) AS totalNeighbors """
pairs = [{" node1 " : fila [ "node1" ], "node2" : fila [ "node2" ] }
para la fila en los datos . recoger ()] características = chispa . createDataFrame ( graph . run ( consulta , { "pairs" : pairs }) . to_data_frame () ) devuelve datos . unirse ( características , [ "node1" , "node2" ] )
```

Y la siguiente función los calculará para el conjunto de prueba:

```
def apply_graphy_test_features ( data ): query = """ UNWIND $ pairs AS pair MATCH (p1) WHERE id (p1) = pair.node1 MATCH
(p2) WHERE id (p2) = pair.node2 RETURN pair.node1 AS node1, pair .node2 AS node2, tamaño ([(p1) - [: CO_AUTHOR] - (a) - [: CO_AUTHOR] - (p2) | a]) AS commonAuthors, tamaño ((p1) - [: CO_AUTHOR] - ()) * tamaño ((p2) - [: CO_AUTHOR] - ()) AS
prefAttachment, tamaño (apoc.coll.toSet ( [(p1) - [: CO_AUTHOR] - (a) | id (a)] + [(p2) - [: CO_AUTHOR] - (a) | id (a)] )) AS
totalNeighbors """
pairs = [{" node1 " : fila [ "node1" ],
"node2" : fila [ "node2" ] }
para la fila en los datos . recoger ()] características = chispa . createDataFrame ( graph . run ( consulta , { "pairs" : pairs }) . to_data_frame () ) devuelve datos . unirse ( características , [ "node1" , "node2" ] )
```

Ambas funciones incorporan un DataFrame que contiene pares de nodos en las columnas node1 y node2 . Luego construimos una serie de mapas que contienen estos pares y calculamos cada una de las medidas para cada par de nodos.

## NOTA

La cláusula UNWIND es particularmente útil en este capítulo para tomar una gran colección de pares de nodos y devolver todas sus características en una consulta.

Podemos aplicar estas funciones en Spark a nuestro entrenamiento y probar los DataFrames con el siguiente código:

```
training_data = apply_graphy_training_features ( training_data ) test_data = apply_graphy_test_features ( test_data )
```

Vamos a explorar los datos en nuestro conjunto de entrenamiento. El siguiente código trazará un histograma de la frecuencia de commonAuthors :

```
PLT . estilo . use ( 'fivethirtyeight' ) fig , axs = plt . subplots ( 1 , 2 , figsize = ( 18 , 7 ) , sharey = True ) charts = [ ( 1 , "have collaborated" ) , ( 0 , "haven't collaborated" )] para índice , chart in enumerate ( charts ) : etiqueta , título = gráfico filtrado = training_data . filtro ( training_data [ "Etiqueta" ] == etiqueta ) common_authors = filtraron . toPandas () [ "commonAuthors" ] histogram = common_authors . value_counts () . sort_index () histogram / = float ( histogram . sum () ) histogram . parcela ( especie = "barra" , x = 'Autores comunes' , color = "darkblue" , ax = axs [ index ] , title = f "Autores que {title} (label = {label})" ) axs [ index ] . xaxis . set_label_text ( "Common Authors" ) plt . tight_layout () plt . mostrar ()
```

Podemos ver el cuadro generado en la [figura 8-8](#) .

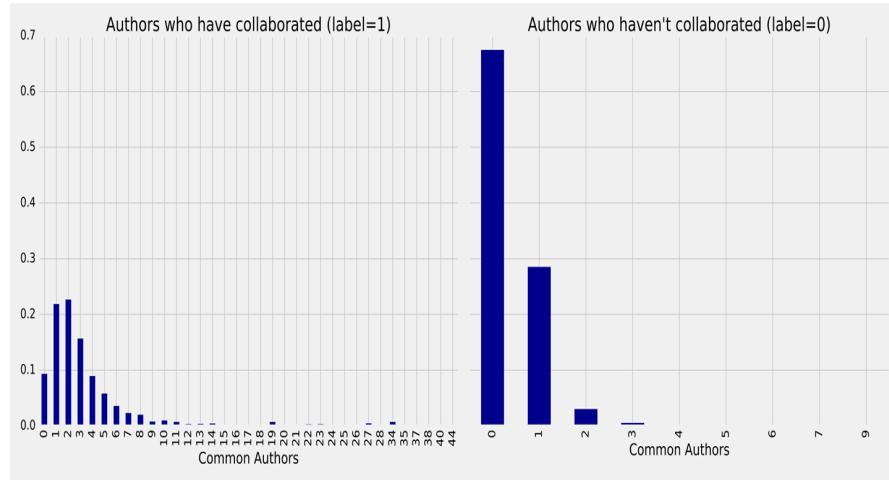


Figura 8-8. Frecuencia de los autores comunes

A la izquierda vemos la frecuencia de los Autores comunes cuando los autores han colaborado, y a la derecha vemos la frecuencia de los Autores comunes cuando no lo han hecho. Para aquellos que no han colaborado (lado derecho), el número máximo de autores comunes es 9, pero el 95% de los valores son 1 o 0. No es sorprendente que de las personas que no han colaborado en un artículo, la mayoría tampoco lo hagan. tienen muchos otros coautores en común. Para aquellos que han colaborado (lado izquierdo), el 70% tiene menos de cinco coautores en común, con un pico entre uno y otros dos coautores.

Ahora queremos entrenar un modelo para predecir enlaces faltantes. La siguiente función hace esto:

```
def train_model ( fields , training_data ): pipeline = create_pipeline ( fields ) model = pipeline . ajuste ( training_data ) modelo de retorno
```

Comenzaremos por crear un modelo básico que solo utilice CommonAuthors . Podemos crear ese modelo ejecutando este código:

```
basic_model = train_model ([ "commonAuthors" ] , training_data )
```

Con nuestro modelo entrenado, vamos a ver cómo funciona contra algunos datos ficticios. El siguiente código evalúa el código con diferentes valores para commonAuthors :

```
eval_df = chispa . createDataFrame ( [ ( 0 ,) , ( 1 ,) , ( 2 ,) , ( 10 ,) , ( 100 ,) ] , [ 'commonAuthors' ] ) ( basic_model . transform ( eval_df ) . select ( "commonAuthors" , "probabilidad" , "predicción" ) . show ( truncate = False ))
```

Ejecutar ese código dará el siguiente resultado:

Autores comunes	probabilidad	predicción
0	[0.7540494940434322,0.24595050595656787]	0.0
1	[0.7540494940434322,0.24595050595656787]	0.0
2	[0.0536835525078107,0.9463164474921892]	1.0
10	[0.0536835525078107,0.9463164474921892]	1.0

Si tenemos un valor de CommonAuthors inferior a 2, hay un 75% de probabilidad de que no haya una relación entre los autores, por lo que nuestro modelo predice 0. Si tenemos un valor de commonAuthors de 2 o más, hay un 94% de probabilidad de que haya. Será una relación entre los autores, por lo que nuestro modelo predice 1.

Ahora evaluemos nuestro modelo contra el conjunto de pruebas. Aunque hay varias formas de evaluar el desempeño de un modelo, la mayoría se derivan de unas pocas métricas predictivas de referencia, como se describe en la [Tabla 8-1](#):

Tabla 8-1. Métricas predictivas

Medida	Fórmula	Descripción
Exactitud	$\frac{\text{TruePositives} + \text{TrueNegatives}}{\text{TotalPredictions}}$	La fracción de predicciones que nuestro modelo hace correcto, o el número total de predicciones correctas dividido por el número total de predicciones. Tenga en cuenta que la precisión por sí sola puede ser engañosa, especialmente cuando nuestros datos no están equilibrados. Por ejemplo, si tenemos un conjunto de datos que contiene 95 gatos y 5 perros y nuestro modelo predice que cada imagen es un gato, tendremos una puntuación del 95% a pesar de no identificar correctamente a ninguno de los perros.
Precisión	$\frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}}$	La proporción de identificaciones positivas que son correctas. Una puntuación de baja precisión indica más falsos positivos. Un modelo que no produce falsos positivos tiene una precisión de 1.0.
Recall (tasa positiva verdadera)	$\frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}}$	La proporción de positivos reales que se identifican correctamente. Un puntaje bajo de recuerdo indica más falsos negativos. Un modelo que no produce falsos negativos tiene un recuerdo de 1.0.
Tasa de falsos positivos	$\frac{\text{FalsePositives}}{\text{FalsePositives} + \text{TrueNegatives}}$	La proporción de positivos incorrectos que se identifican. Una puntuación alta indica más falsos positivos.
Característica de funcionamiento Carta XY (ROC) curva		La curva ROC es una gráfica del Recall (tasa verdadera positiva) contra la tasa False Positive en diferentes umbrales de clasificación. El área debajo de la curva (AUC) mide el área bidimensional debajo de la curva ROC desde un eje XY (0,0) a (1,1).

Usaremos la precisión, la recuperación y las curvas ROC para evaluar nuestros modelos. La precisión es una medida general, por lo que nos centraremos en aumentar nuestra precisión general y las medidas de recuperación. Usaremos las curvas ROC para comparar cómo las características individuales cambian las tasas predictivas.

## PROPIÑA

Dependiendo de nuestras metas podemos querer favorecer diferentes medidas. Por ejemplo, podríamos querer eliminar todos los falsos negativos para los indicadores de enfermedad, pero no queríamos que las predicciones de todo lleguen a un resultado positivo. Puede haber múltiples umbrales que configuramos para diferentes modelos que pasan algunos resultados a una inspección secundaria sobre la probabilidad de resultados falsos.

La reducción de los umbrales de clasificación da como resultado resultados positivos más generales, lo que aumenta tanto los falsos positivos como los verdaderos positivos.

Usemos la siguiente función para calcular estas medidas predictivas:

```
def evaluate_model ( modelo , test_data ): # Ejecutar el modelo contra el conjunto de prueba
    predicciones = modelo . transformar ( test_data ) # Calcular verdaderos, falsos positivos, falsos negativos, recuentos
    tp = predicciones [ ( predicciones . etiqueta == 1 ) y ( predicciones . predicción == 1 ) ] . cuenta () fp = predicciones [ ( predicciones . label == 0 ) & ( predictions . prediction == 1 ) ] . count () fn = predicciones [ ( predictions . label == 1 ) & ( predictions . prediction == 0 ) ] . contar () de llamada
```

```

número Compute y precisión manualmente recuerdan = float ( tp ) / ( TP + fn ) de precisión =
flotador ( tp ) / ( tp + fp ) exactitud # Compute utilizando la clasificación evaluador binario Spark de MLLib exactitud =
BinaryClassificationEvaluator () . evaluar ( predicciones ) # Calcular la tasa de falsos positivos y la tasa de verdaderos positivos usando
las funciones sklearn labels = [ row [ "label" ] para la fila en las predicciones . seleccione ( "etiqueta" ) . collect ()] preds =
[ fila [ "probabilidad" ] [ 1 ] para fila en predicciones . seleccionar ( "probabilidad" ) . recoger ()] FPR , tpr , umbral = roc_curve (
etiquetas , Preds ) roc_auc = AUC ( FPR , TPR ) de retorno { "FPR" : FPR , "tpr" : tpr , "roc_auc" :
"exactitud" : precisión , "recordar" : recordar , "precisión" : precisión }

```

Luego, escribiremos una función para mostrar los resultados en un formato más fácil de consumir:

```

def display_results ( results ): results = { k : v para k , v en results . items () si k no está en [ "fpr" , "tpr" , "roc_auc" ]} return pd .
DataFrame ({ "Measure" : list ( results . Keys () ), "Score" : list ( results . Valores
() )})

```

Podemos llamar a la función con este código y mostrar los resultados:

```
basic_results = evaluate_model ( basic_model , test_data ) display_results ( basic_results )
```

Las medidas predictivas para el modelo de autores comunes son:

medida Puntuación

exactitud 0.864457

recordar 0.753278

precisión 0.968670

Este no es un mal comienzo dado que estamos pronosticando futuras colaboraciones basadas solo en el número de autores comunes en nuestros pares de autores. Sin embargo, tenemos una imagen más amplia si consideramos estas medidas en contexto entre sí. Por ejemplo, este modelo tiene una precisión de 0.968670, lo que significa que es muy bueno para predecir que los enlaces existen. Sin embargo, nuestro retiro es 0.753278, lo que significa que no es bueno para predecir cuándo no existen los enlaces.

También podemos trazar la curva ROC (correlación de verdaderos positivos y falsos positivos) utilizando las siguientes funciones:

```

def create_roc_plot (): plt . estilo . utilizar ( 'clásico' ) fig = plt . figura ( figsize = ( 13 , 8 )) plt . xlim ([ 0 , 1 ]) plt . ylim ([ 0 , 1 ]) plt .
ylabel ( 'True Positive Rate' ) plt . xlabel ( 'tasa de falsos positivos' ) PLT . rc ( 'ejes' , prop_cycle = ( ciclador (
'color' , [ 'r' , 'g' , 'b' , 'c' , 'm' , 'y' , 'k' ])) plt . gráfico ([ 0 , 1 ], [ 0 , 1 ], estilo de línea = '-' , etiqueta = 'Puntuación
aleatoria ( AUC = 0. aleatoria plt . def add_curve ( plt , título , FPR , tpr , ROC ): plt . parcela ( FPR , tpr , etiqueta = f "{title}
(AUC = {roc: 0,2})" )

```

Lo llamamos así:

```
plt , fig = create_roc_plot () add_curve ( plt , "Common Authors" , basic_results [ "fpr" ], basic_results [ "tpr" ], basic_results [
"roc_auc" ]) plt . leyenda ( loc = 'abajo a la derecha' ) plt . mostrar ()

```

Podemos ver la curva ROC para nuestro modelo básico en la [Figura 8-9](#).

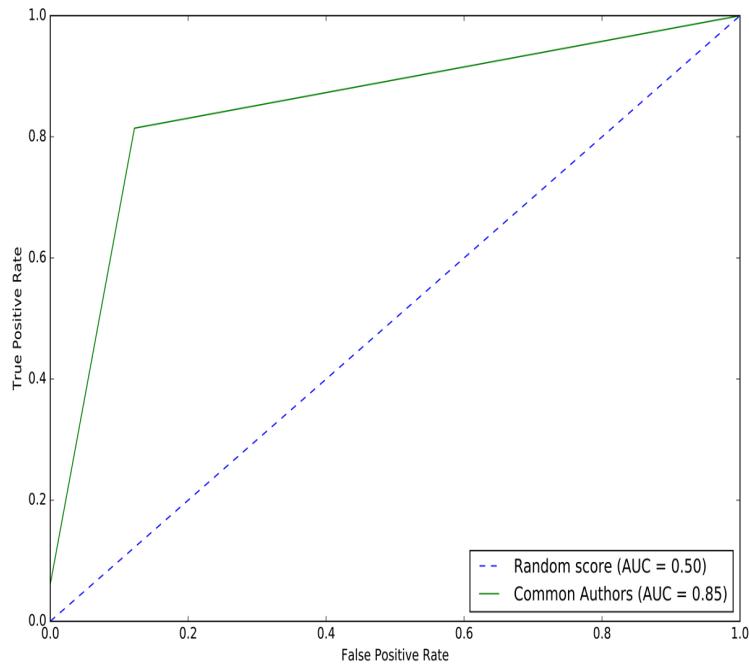


Figura 8-9. La curva ROC para el modelo básico.

El modelo de autores comunes nos da una puntuación de área bajo la curva (AUC) de 0.86. Aunque esto nos da una medida predictiva general, necesitamos la tabla (u otras medidas) para evaluar si esto se ajusta a nuestra meta. En la [Figura 8-9](#) vemos que a medida que nos acercamos a una tasa de verdaderos positivos del 80% (recordemos) nuestra tasa de falsos positivos alcanza aproximadamente el 20%. Eso podría ser problemático en escenarios como la detección de fraudes donde los falsos positivos son caros de perseguir. Ahora usemos las otras características de grafía para ver si podemos mejorar nuestras predicciones. Antes de entrenar nuestro modelo, veamos cómo se distribuyen los datos. Podemos ejecutar el siguiente código para mostrar estadísticas descriptivas de cada una de nuestras funciones de grafía:

```
( training_data . filter ( training_data [ "label" ] == 1 ) . describe () . select ( "summary" , "commonAuthors" , "prefAttachment" ,
"totalNeighbors" ) . show () )

( training_data . filter ( training_data [ "label" ] == 0 ) . describe () . select ( "summary" , "commonAuthors" , "prefAttachment" ,
"totalNeighbors" ) . show () )
```

Podemos ver los resultados de ejecutar esos bits de código en las siguientes tablas:

<b>resumen</b>	<b>Autores comunes</b>	<b>pre-adjunto</b>	<b>TotalNeighbors</b>
contar	81096	81096	81096
media	3.5959233501035808	69.93537289138798	10.082408503502021
stddev	4.715942231635516	171.47092255919472	8.44109970920685
min	0	1	2
max	44	3150	90
<b>resumen</b>	<b>Autores comunes</b>	<b>pre-adjunto</b>	<b>TotalNeighbors</b>
contar	81096	81096	81096
media	0.37666469369635985	48.18137762651672	12.97586810693499
stddev	0.6194576095461857	94.92635344980489	10.082991078685803
min	0	1	1
max	9	1849	89

Las características con diferencias más grandes entre enlaces (coautoría) y ningún enlace (sin coautoría) deberían ser más predictivas porque la división es mayor. El valor promedio de prefAttachment es mayor para los autores que han colaborado en comparación con aquellos que no lo han hecho. Esa diferencia es aún más sustancial para los autores comunes . Notamos que no hay mucha diferencia en los valores para el total de vecinos , lo que probablemente significa que esta característica no será muy predictiva. También es interesante la gran desviación estándar, así como los valores mínimos y máximos para la conexión preferencial. Esto es lo que podríamos esperar para las redes del mundo pequeño con concentradores concentrados (superconectores).

Ahora entrenemos un nuevo modelo, agregando un adjunto preferencial y una unión total de vecinos, ejecutando el siguiente código:

```
fields = [ "commonAuthors" , "prefAttachment" , "totalNeighbors" ] graphy_model = train_model ( fields , training_data )
```

Y ahora vamos a evaluar el modelo y mostrar los resultados:

```
graphy_results = evaluate_model ( graphy_model , test_data ) display_results ( graphy_results )
```

Las medidas predictivas para el modelo de grafía son:

medida Puntuación

exactitud 0.978351

recordar 0.924226

precisión 0.943795

Nuestra precisión y recuperación han aumentado sustancialmente, pero la precisión ha disminuido un poco y todavía estamos clasificando incorrectamente alrededor del 8% de los enlaces. Vamos a trazar la curva ROC y comparar nuestros modelos básicos y de grafía ejecutando el siguiente código:

```
plt , fig = create_roc_plot () add_curve ( plt , "Common Authors" , basic_results [ "fpr" ], basic_results [ "tpr" ], basic_results [ "roc_auc" ]) add_curve ( plt , " Graphy " , graphy_results [ "fpr" ], graphy_results [ "tpr" ], graphy_results [ "roc_auc" ]) plt .leyenda ( loc = 'abajo a la derecha' ) plt .mostrar ()
```

Podemos ver la salida en la [Figura 8-10](#) .

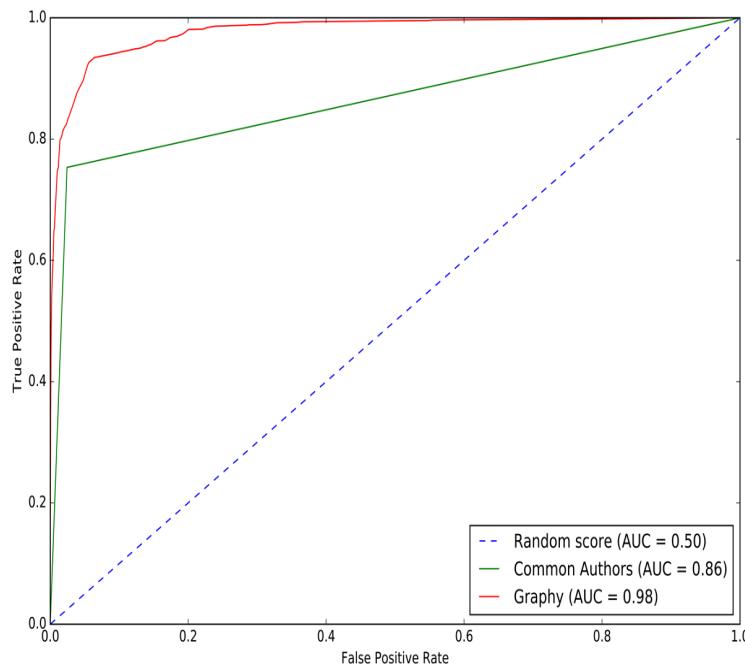


Figura 8-10. La curva ROC para el modelo de grafía.

En general, parece que nos dirigimos en la dirección correcta y es útil visualizar las comparaciones para tener una idea de cómo los diferentes modelos impactan nuestros resultados.

Ahora que tenemos más de una característica, queremos evaluar qué características hacen la mayor diferencia. Usaremos la importancia de las características para clasificar el impacto de las diferentes características según la predicción de nuestro modelo. Esto nos permite evaluar la influencia en los resultados que tienen los diferentes algoritmos y estadísticas.

### NOTA

Para calcular la importancia de las características, el algoritmo de bosque aleatorio en Spark promedia la reducción de impurezas en todos los árboles del bosque. La impureza es la frecuencia con la que las etiquetas asignadas al azar son incorrectas.

Las clasificaciones de características se comparan con el grupo de características que estamos evaluando, siempre normalizadas a 1. Si clasificamos una característica, su importancia es 1.0, ya que tiene el 100% de la influencia en el modelo.

La siguiente función crea un gráfico que muestra las características más influyentes:

```
def plot_feature_importance ( fields , feature_importances ): df = pd . Marco de datos ( { "Característica" : campos , "Importancia" : feature_importances } ) df = df . sort_values ( "Importancia" , ascendente = Falso ) ax = df . plot ( kind = 'bar' , x = 'Feature' , y = 'Importancia' , leyenda = Ninguna ) ax . xaxis . set_label_text ( "" ) plt . tight_layout () plt . mostrar ()
```

Y lo llamamos así:

```
rf_model = graphy_model . etapas [ - 1 ] plot_feature_importance ( campos , rf_model . featureImportances )
```

Los resultados de ejecutar esa función se pueden ver en la [Figura 8-11](#) .

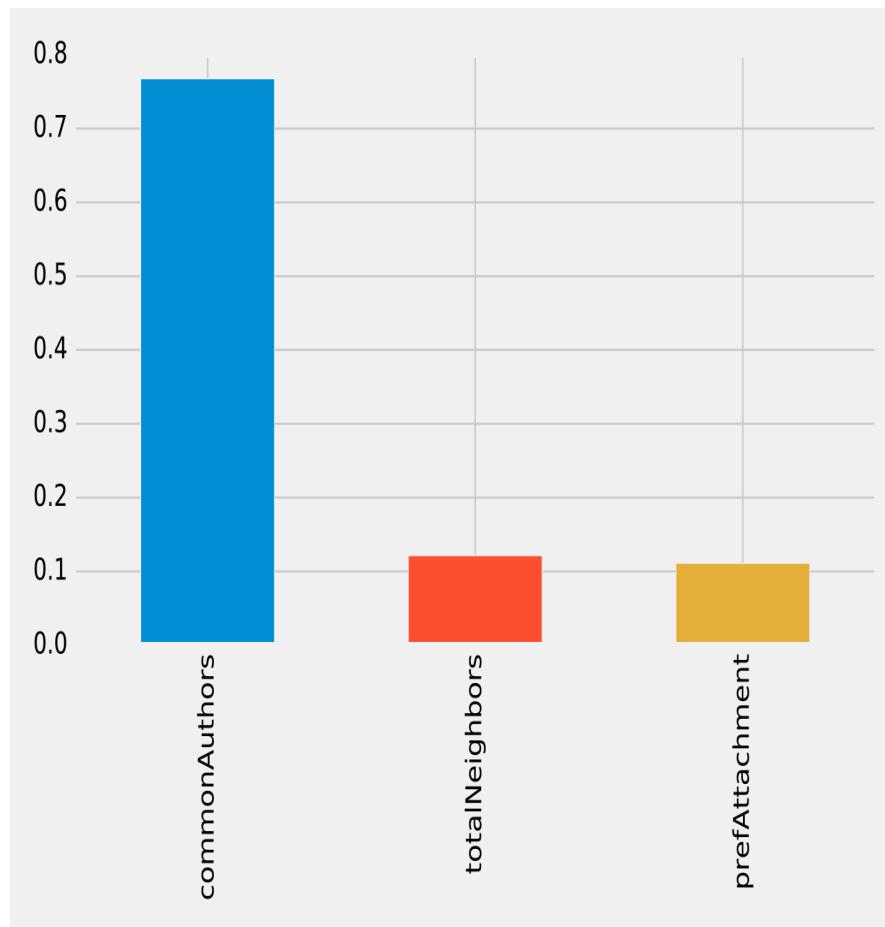


Figura 8-11. Característica importante: modelo de graña

De las tres características que hemos usado hasta ahora, commonAuthors es la característica más importante por un amplio margen.

Para comprender cómo se crean nuestros modelos predictivos, podemos visualizar uno de los árboles de decisión en nuestro bosque aleatorio utilizando la [biblioteca de trazado de árboles de chispas](#) . El siguiente código genera un [archivo GraphViz](#) :

```
desde spark_tree_plotting import export_graphviz
dot_string = export_graphviz ( rf_model . trees [ 0 ], featureNames = fields ,
categoryNames = [], classNames = [ "True" , "False" ], relleno = True , roundedCorners = True , roundLeaves = True ) with open (
"/tmp/rf.dot" , "w" ) como archivo : archivo . escribir ( dot_string )
```

Luego podemos generar una representación visual de ese archivo ejecutando el siguiente comando desde el terminal:

```
punto -Tpdf /tmp/rf.dot -o /tmp/rf.pdf
```

La salida de ese comando se puede ver en la [Figura 8-12](#) .

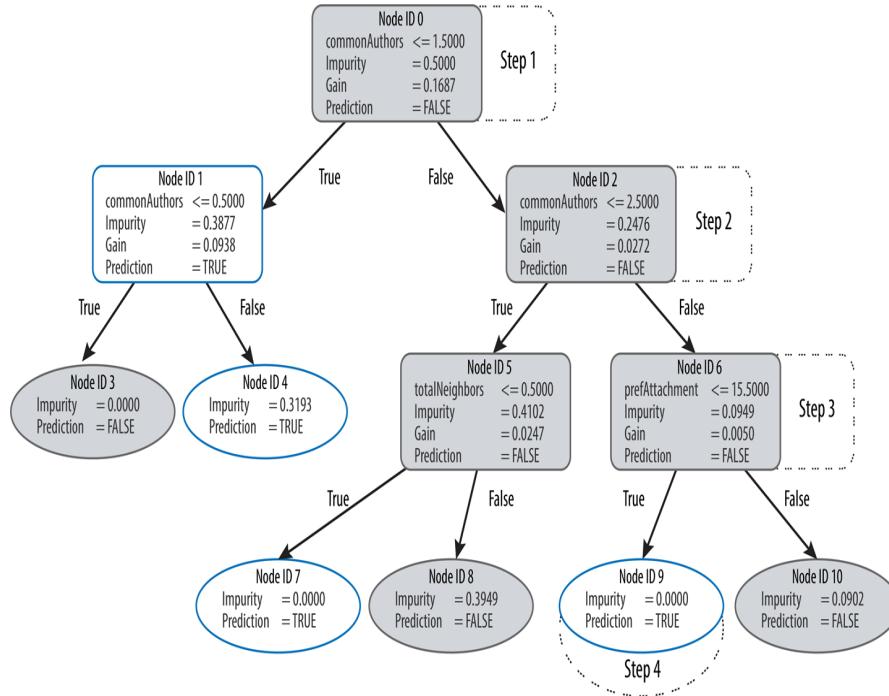


Figura 8-12. Visualizando un árbol de decisión

Imagine que estamos usando este árbol de decisión para predecir si un par de nodos con las siguientes funciones están vinculados:

#### **Autores comunes pre-adjunto TotalNeighbors**

10	12	5
----	----	---

Nuestro bosque aleatorio recorre varios pasos para crear una predicción:

1. Comenzamos desde el nodo 0 , donde tenemos más de 1.5 CommonAuthors , por lo que seguimos la rama falsa hasta el nodo 2 .
2. Tenemos más de 2.5 CommonAuthors aquí, por lo que seguimos la rama falsa al nodo 6 .
3. Tenemos una puntuación de menos de 15.5 para prefAttachment , que nos lleva al nodo 9 .
4. El nodo 9 es un nodo de hoja en este árbol de decisión, lo que significa que no tenemos que verificar más condiciones: el valor de Predicción (es decir, Verdadero ) en este nodo es la predicción del árbol de decisión.
5. Finalmente, el bosque aleatorio evalúa el elemento que se está pronosticando contra una colección de estos árboles de decisión y hace su predicción basada en el resultado más popular.

Ahora veamos cómo agregar más características gráficas.

## **Predicción de enlaces: triángulos y el coeficiente de agrupación**

Las soluciones de recomendación a menudo basan las predicciones en alguna forma de métrica de triángulo, así que veamos si ayudan más con nuestro ejemplo. Podemos calcular la cantidad de triángulos de los que forma parte un nodo y su coeficiente de agrupamiento ejecutando la siguiente consulta:

```
CALL algo.triangleCount ('Autor', 'CO_AUTHOR_EARLY', {write: true , writeProperty: 'trianglesTrain' ,
clusteringCoefficientProperty: 'coefficientTrain' }); CALL algo.triangleCount ('Autor', 'CO_AUTHOR', {write: true , writeProperty:
'trianglesTest' , clusteringCoefficientProperty: 'coefficientTest' });
```

La siguiente función agregará estas características a nuestros DataFrames:

```
def apply_triangles_features ( data , triangles_prop , coefficient_prop ): query = " " " UNWIND $ pairs AS pair MATCH ( p1 ) WHERE
id ( p1 ) = pair . node1 MATCH ( p2 ) WHERE id ( p2 ) = pair . node2 RETURN pair . node1 AS node1 , pair . node2 AS node2 , apoc . coll . min
([p1 [ $ trianglesProp ] , p2 [ $ trianglesProp ]]) AS minTriangles , apoc . coll . max ([p1 [ $ trianglesProp ] , p2 [ $ trianglesProp ]]) AS
maxTriangles , apoc . coll . min ([p1 [ $ coefficientProp ] , p2 [ $ coefficientProp ]]) AS minCoefficient , apoc . coll . max ([p1 [ $ coefficientProp ] , p2 [ $ coefficientProp ]]) AS maxCoefficient " "
params = { "pairs" : [
"node1" : row [ "node1" ] , "node2" : row [ "node2" ] } para la fila en los datos . collect () , "trianglesProp" : triangles_prop ,
"coefficientProp" : coefficient_prop } features = spark . createDataFrame ( graph . run ( query , params ) .
() Devuelve datos . unirse ( características , [ "node1" , "node2" ] )
```

## NOTA

Tenga en cuenta que hemos utilizado los prefijos mínimo y máximo para nuestros algoritmos de coeficiente de agrupación y triángulo. Necesitamos una forma de evitar que nuestro modelo aprenda según el orden en que los autores se transmiten en pares desde nuestro gráfico no dirigido. Para ello, hemos dividido estas características por los autores con conteos mínimos y máximos.

Podemos aplicar esta función a nuestro entrenamiento y probar DataFrames con el siguiente código:

```
training_data = apply_triangles_features ( training_data , "trianglesTrain" , "coefficientTrain" ) test_data = apply_triangles_features (
test_data , "trianglesTest" , "coefficientTest" )
```

Y ejecute este código para mostrar estadísticas descriptivas de cada una de nuestras características de triángulo:

```
( training_data . filter ( training_data [ "label" ] == 1 ) . describe () . select ( "summary" , "minTriangles" , "maxTriangles" ,
"minCoefficient" , "maxCoefficient" ) . show () )
( training_data . filter ( training_data [ "label" ] == 0 ) . describe () . select ( "summary" , "minTriangles" , "maxTriangles" ,
"minCoefficient" , "maxCoefficient" ) . show () )
```

Podemos ver los resultados de ejecutar esos bits de código en las siguientes tablas.

<b>resumen</b>	<b>minTriángulos</b>	<b>maxTriangles</b>	<b>minCoeficiente</b>	<b>maxCoefficient</b>
contar	81096	81096	81096	81096
media	19.478260333431983	27.73590559337082	0.5703773654487051	0.8453786164620439
stddev	65.7615282768483	74.01896188921927	0.3614610553659958	0.2939681857356519
min	0	0	0.0	0.0
max	622	785	1.0	1.0
<b>resumen</b>	<b>minTriángulos</b>	<b>maxTriangles</b>	<b>minCoeficiente</b>	<b>maxCoefficient</b>
contar	81096	81096	81096	81096
media	5.754661142349808	35.651980368945445	0.49048921333297446	0.860283935358397
stddev	20.639236521699	85.82843448272624	0.3684138346533951	0.2578219623967906
min	0	0	0.0	0.0
max	617	785	1.0	1.0

Observe en esta comparación que no hay una gran diferencia entre los datos de coautoría y no coautor. Esto podría significar que estas características no son tan predictivas.

Podemos entrenar otro modelo ejecutando el siguiente código:

```
fields = [ "commonAuthors" , "prefAttachment" , "totalNeighbors" , "minTriangles" , "maxTriangles" , "minCoefficient" ,
"maxCoefficient" ] triangle_model = train_model ( fields , training_data )
```

Y ahora vamos a evaluar el modelo y mostrar los resultados:

```
triangle_results = evaluate_model ( triangle_model , test_data ) display_results ( triangle_results )
```

Las medidas predictivas para el modelo de triángulos se muestran en esta tabla:

medida	Puntuación
--------	------------

exactitud	0.992924
-----------	----------

recordar	0.965384
----------	----------

precisión	0.958582
-----------	----------

Nuestras medidas predictivas han aumentado bien al agregar cada nueva característica al modelo anterior.

Agreguemos nuestro modelo de triángulos a nuestro gráfico de curvas ROC con el siguiente código:

```
plt , fig = create_roc_plot () add_curve ( plt , "Common Authors" , basic_results [ "fpr" ], basic_results [ "tpr" ], basic_results [ "roc_auc" ]) add_curve ( plt , "Graphy" , graphy_results [ "fpr" ], graphy_results [ "tpr" ], graphy_results [ "roc_auc" ]) add_curve ( plt , "Triangulos" , triangle_results [ "fpr" ], triangle_results [ "tpr" ], triangle_results [ "roc_auc" ]) plt . leyenda ( loc = 'abajo a la derecha' ) plt . mostrar ()
```

Podemos ver la salida en la [Figura 8-13](#).

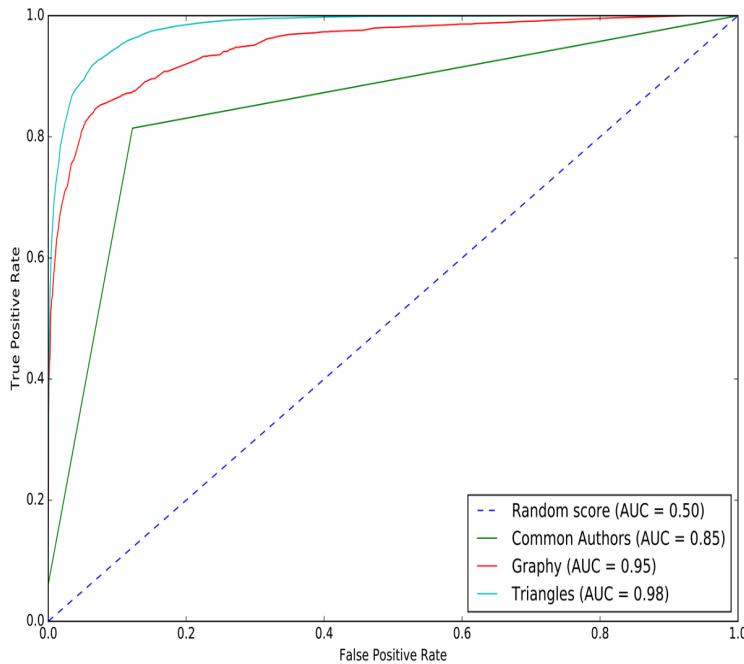


Figura 8-13. La curva ROC para el modelo de triángulos.

Nuestros modelos en general han mejorado, y estamos en los 90 altos para las medidas predictivas. Esto es cuando las cosas generalmente se ponen difíciles, porque se obtienen las ganancias más fáciles, pero todavía hay espacio para mejorar. Veamos cómo han cambiado las características importantes:

```
rf_model = triangle_model . etapas [ - 1 ] plot_feature_importance ( campos , rf_model . featureImportances )
```

Los resultados de ejecutar esa función se pueden ver en la [Figura 8-14](#).

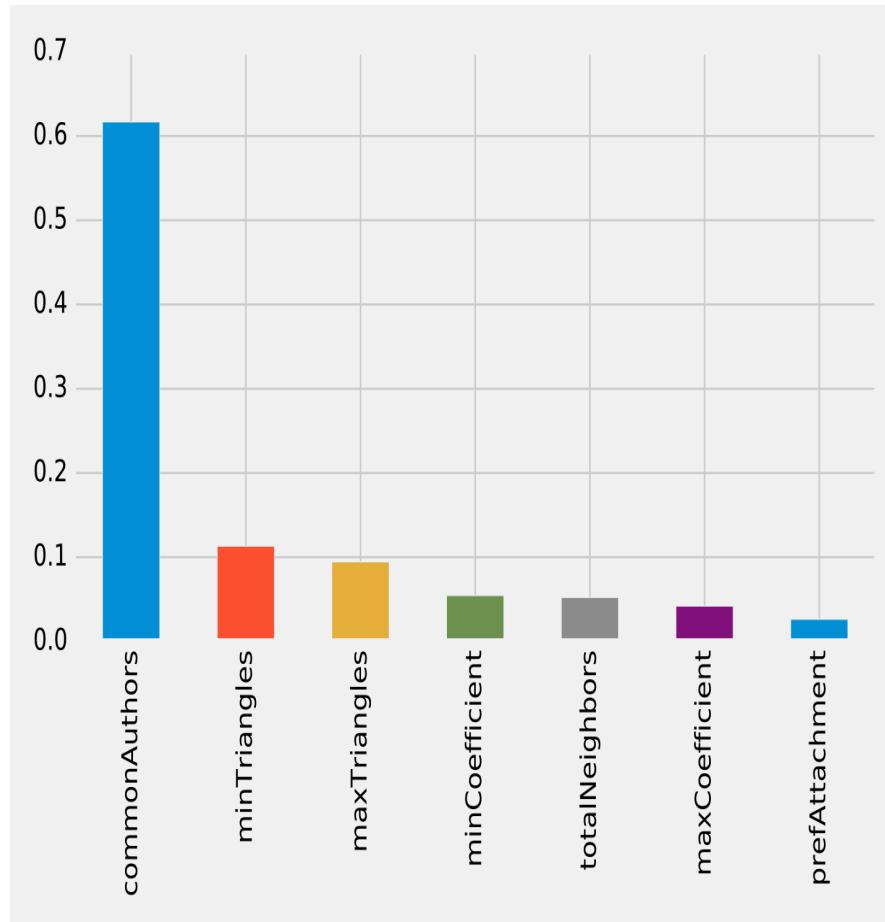


Figura 8-14. Característica importante: modelo de triángulos

La característica de autores comunes todavía tiene el mayor impacto individual en nuestro modelo. Tal vez necesitamos ver nuevas áreas y ver qué sucede cuando agregamos información de la comunidad.

## Predicción de enlaces: Detección comunitaria

Nuestra hipótesis es que los nodos que están en la misma comunidad tienen más probabilidades de tener un vínculo entre ellos si no lo hacen ya. Además, creemos que cuanto más estrecha es una comunidad, más probable es que haya vínculos.

Primero, calcularemos comunidades más complejas utilizando el algoritmo de propagación de etiquetas en Neo4j. Hacemos esto ejecutando la siguiente consulta, que almacenará la comunidad en la propiedad partitionTrain para el conjunto de entrenamiento y partitionTest para el conjunto de prueba:

```
CALL algo.labelPropagation ("Autor", "CO_AUTHOR_EARLY", "BOTH", {partitionProperty: "partitionTrain"}); CALL algo.labelPropagation ("Autor", "CO_AUTHOR", "BOTH", {partitionProperty: "partitionTest"});
```

También calcularemos grupos más finos utilizando el algoritmo de Lovaina. El algoritmo de Louvain devuelve clústeres intermedios, y almacenaremos el más pequeño de estos clústeres en la propiedad louvainTrain para el conjunto de entrenamiento y louvainTest para el conjunto de prueba:

```
CALL algo.louvain.stream ("Author", "CO_AUTHOR_EARLY", {includeIntermediateCommunities: true}) nodeId de YIELD, comunidad, comunidades WITH algo.getNodeById (nodeId) AS node , comunidades [0] AS smallestCommunity SET node .louvainTrain = smallestCommunity; CALL algo.louvain.stream ("Author", "CO_AUTHOR", {includeIntermediateCommunities: true}) ID de nodo, comunidad, comunidades CON algo.getNodeById (nodeId) AS nodo , comunidades [0] AS smallestCommunity SET node .louvainTest = smallestCommunity;
```

Ahora crearemos la siguiente función para devolver los valores de estos algoritmos:

```
def apply_community_features ( data , partition_prop , louvain_prop ): query = " UNWIND $ pairs AS pair MATCH (p1) WHERE id (p1) = pair.node1 MATCH (p2) WHERE id (p2) = pair.node2 RETURN pair.node1 AS node1 , pair.node2 AS node2, CASE WHEN p1 [$ partitionProp] = p2 [$ partitionProp] THEN 1 ELSE 0 FIN COMO samePartition, CASE WHEN p1 [$ louvainProp] = p2 [$ louvainProp] THEN 1 ELSE 0 END AS sameLouvain " " params = { " pairs " : [{ " node1 " : fila [ " node1 " ] "node2" : fila [ "node2" ]} para la fila en los datos . collect ()], "partitionProp" :
```

```
partition_prop , "louvainProp" : louvain_prop } features = spark . createDataFrame ( graph . run ( query , params ) . to_data_frame () )
devuelve datos . unirse ( características , [ "node1" , "node2" ]
])
```

Podemos aplicar esta función a nuestro entrenamiento y probar DataFrames en Spark con el siguiente código:

```
training_data = apply_community_features ( training_data , "partitionTrain" , "louvainTrain" ) test_data = apply_community_features ( test_data , "partitionTest" , "louvainTest" )
```

Y podemos ejecutar este código para ver si los pares de nodos pertenecen a la misma partición:

```
PLT . estilo . use ( 'fivethirtyeight' ) fig , axs = plt . subplots ( 1 , 2 , figsize = ( 18 , 7 ) , sharey = True ) charts = [ ( 1 , "have collaborated" ), ( 0 , "have collaborated" ) ] para índice , chart in enumerate ( charts ) : etiqueta , título = gráfico filtrado = training_data . filtro ( training_data [ "Etiqueta" ] == etiqueta ) valores = ( filtrada . withColumn ( 'samePartition' , F . col ( "samePartition" ) == 0 , "False" ) . de otro modo ( "True" ) ) . groupby ( "samePartition" ) . agg ( F . recuento ( "etiqueta" ) . alias ( "Count" ) ) . seleccione ( "samePartition" , "count" ) . toPandas () . valores . set_index ( "samePartition" , drop = True , inplace = True ) . valores . parcela ( kind = "bar" , ax = axs [ índice ] , leyenda = Ninguno , título = f "Autores que {título} (etiqueta = {etiqueta})" ) . axs [ índice ] . xaxis . set_label_text ( "Same Partition" ) . plt . tight_layout () plt . mostrar ( )
```

Vemos los resultados de ejecutar ese código en la [Figura 8-15](#).

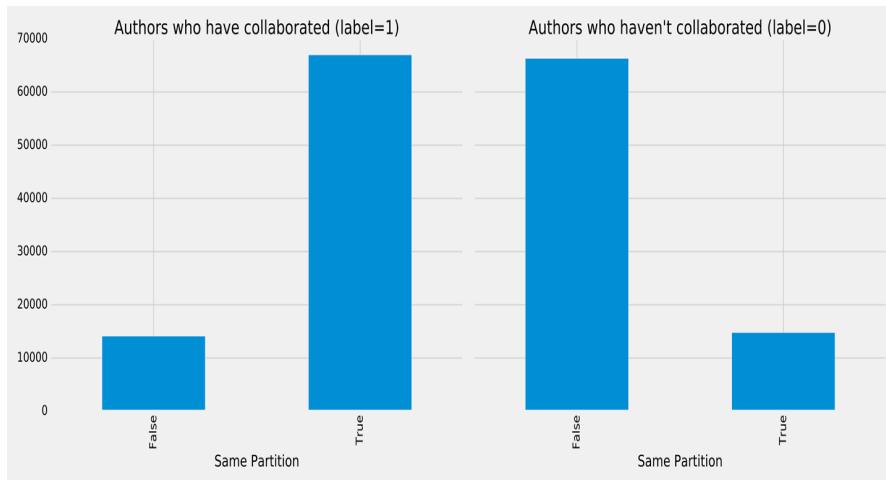


Figura 8-15. Mismas particiones

Parece que esta característica podría ser bastante predictiva: los autores que han colaborado tienen muchas más probabilidades de estar en la misma partición que los que no lo han hecho. Podemos hacer lo mismo para los clusters de Louvain ejecutando el siguiente código:

```
PLT . estilo . use ( 'fivethirtyeight' ) fig , axs = plt . subplots ( 1 , 2 , figsize = ( 18 , 7 ) , sharey = True ) charts = [ ( 1 , "have collaborated" ), ( 0 , "have collaborated" ) ] para índice , chart in enumerate ( charts ) : etiqueta , título = gráfico filtrado = training_data . filtro ( training_data [ "Etiqueta" ] == etiqueta ) valores = ( filtrada . withColumn ( 'sameLouvain' , F . col ( "sameLouvain" ) == 0 , "False" ) . de otro modo ( "True" ) ) . groupby ( "sameLouvain" ) . agg ( F . recuento ( "etiqueta" ) . alias ( "Count" ) ) . seleccione ( "sameLouvain" , "count" ) . toPandas () . valores . Set_index ( "sameLouvain" , drop = True , inplace = True ) . valores . parcela ( kind = "bar" , índice ) , leyenda = Ninguno , título = f "Autores que {título} (etiqueta = {etiqueta})" ) . axs [ índice ] . xaxis . set_label_text ( "Same Louvain" ) . plt . tight_layout () plt . mostrar ( )
```

Podemos ver los resultados de ejecutar ese código en la [Figura 8-16](#).

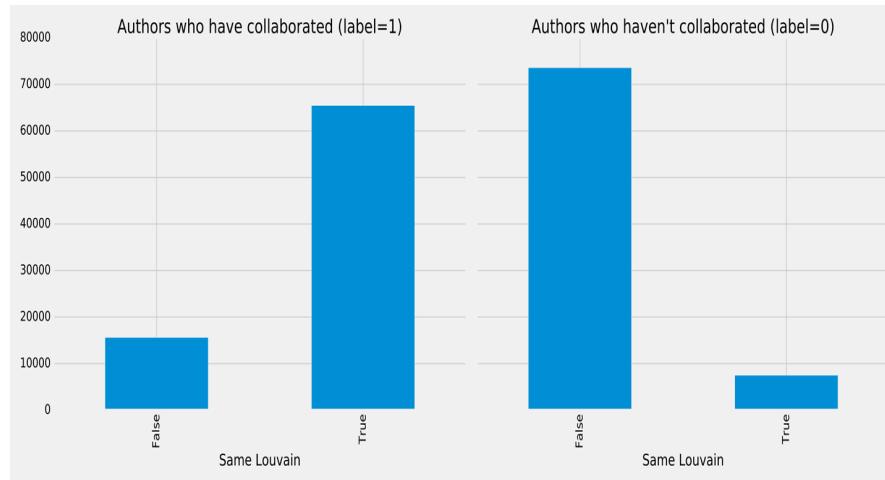


Figura 8-16. Los mismos clusters de Louvain

Parece que esta característica también puede ser bastante predictiva: es probable que los autores que han colaborado estén en el mismo grupo, y es probable que los que no lo hagan estén en el mismo grupo.

Podemos entrenar otro modelo ejecutando el siguiente código:

```
fields = [ "commonAuthors" , "prefAttachment" , "totalNeighbors" , "minTriangles" , "maxTriangles" , "minCoefficient" ,  
"maxCoefficient" , "samePartition" , "sameLouvain" ] community_model = train_model ( fields , training_data )
```

Y ahora vamos a evaluar el modelo y mostrar los resultados:

```
community_results = evaluate_model ( community_model , test_data ) display_results ( community_results )
```

Las medidas predictivas para el modelo de comunidad son:

#### medida Puntuación

exactitud 0.995771

recordar 0.957088

precisión 0.978674

Algunas de nuestras medidas han mejorado, así que para la comparación, vamos a trazar la curva ROC para todos nuestros modelos ejecutando el siguiente código:

```
plt , fig = create_roc_plot () add_curve ( plt , "Common Authors" , basic_results [ "fpr" ] , basic_results [ "tpr" ] , basic_results [  
"roc_auc" ]) add_curve ( plt , "Graphy" , graphy_results [ "fpr" ] , graphy_results [ "tpr" ] , graphy_results [ "roc_auc" ]) add_curve ( plt  
,"Triangulos" , triangle_results [ "fpr" ] , triangle_results [ "tpr" ] , triangle_results [ "roc_auc" ]) add_curve ( plt  
,"Community" , community_results [ "fpr" ] , community_results [ "tpr" ] , community_results [ "roc_auc" ]) plt . leyenda ( loc = 'abajo a la derecha' ) plt . mostrar ()
```

Podemos ver la salida en la [figura 8-17](#).

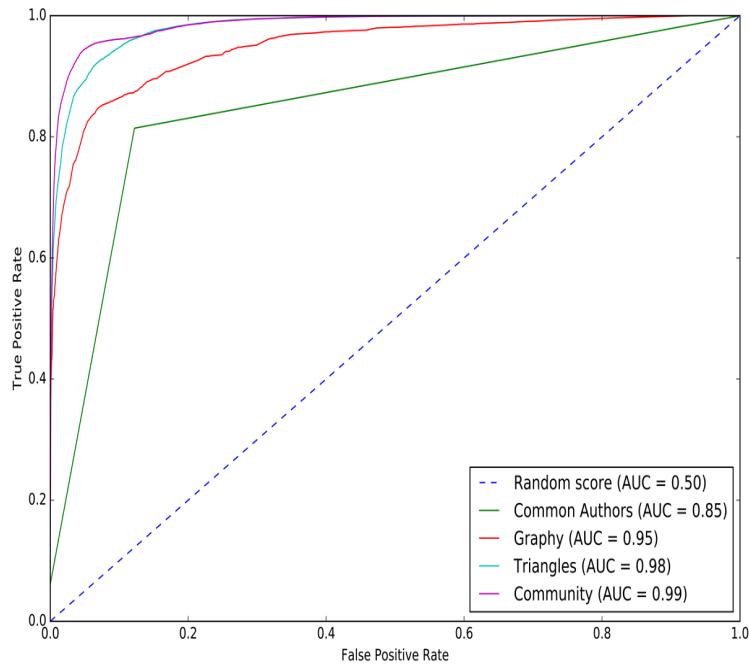


Figura 8-17. La curva ROC para el modelo de comunidad.

Podemos ver mejoras con la adición del modelo de comunidad, así que veamos cuáles son las características más importantes:

```
rf_model = community_model . etapas [ - 1 ] plot_feature_importance ( campos , rf_model . featureImportances )
```

Los resultados de ejecutar esa función se pueden ver en la [Figura 8-18](#) .

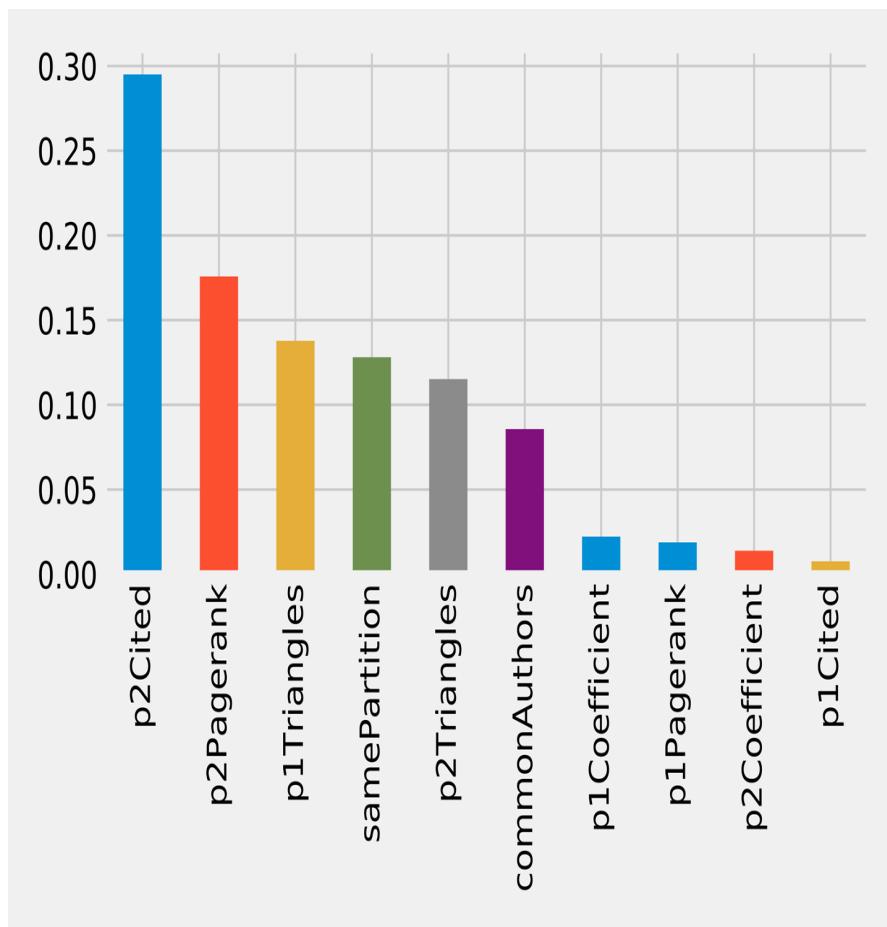


Figura 8-18. Característica importante: modelo de comunidad

Si bien el modelo de autores comunes es en general muy importante, es bueno evitar tener un elemento demasiado dominante que pueda sesgar las predicciones en nuevos datos. Los algoritmos de detección de la comunidad tuvieron mucha influencia en nuestro último modelo con todas las características incluidas, y esto ayuda a completar nuestro enfoque predictivo.

Hemos visto en nuestros ejemplos que las características simples basadas en gráficos son un buen comienzo, y luego, a medida que agregamos más funciones basadas en algoritmos de grafos y gráficas, continuamos mejorando nuestras medidas predictivas. Ahora tenemos un modelo bueno y equilibrado para predecir enlaces de coautor.

El uso de gráficos para la extracción de características conectadas puede mejorar significativamente nuestras predicciones. Las características y algoritmos ideales del gráfico varían según los atributos de los datos, incluidos el dominio de la red y la forma del gráfico. Primero sugerimos considerar los elementos predictivos dentro de sus datos y probar hipótesis con diferentes tipos de funciones conectadas antes de realizar ajustes precisos..

## EJERCICIOS PARA LECTORES

Hay varias áreas para investigar y formas de construir otros modelos. Aquí hay algunas ideas para una mayor exploración:

- ¿Qué tan predictivo es nuestro modelo sobre los datos de la conferencia que no incluimos?
- Al probar nuevos datos, ¿qué sucede cuando eliminamos algunas características?
- ¿La división de los años de manera diferente para la capacitación y las pruebas afecta nuestras predicciones?
- Este conjunto de datos también tiene citas entre los documentos; ¿Podemos usar esos datos para generar características diferentes o predecir citas futuras?

## Resumen

En este capítulo, analizamos el uso de algoritmos y características de gráficos para mejorar el aprendizaje automático. Cubrimos algunos conceptos preliminares y luego analizamos un ejemplo detallado que integra Neo4j y Apache Spark para la predicción de enlaces. Ilustramos cómo evaluar modelos clasificadores de bosques aleatorios e incorporar varios tipos de funciones conectadas para mejorar nuestros resultados.

## Envolviendo las cosas

En este libro, cubrimos los conceptos gráficos, así como las plataformas de procesamiento y análisis. Luego caminamos a través de muchos ejemplos prácticos de cómo usar algoritmos de gráficos en Apache Spark y Neo4j. Terminamos con una mirada a cómo los gráficos mejoran el aprendizaje automático.

Los algoritmos de grafos son la fuente de poder detrás del análisis de los sistemas del mundo real, desde la prevención del fraude y la optimización del enrutamiento de llamadas hasta la predicción de la propagación de la gripe. Esperamos que se una a nosotros y desarrolle sus propias soluciones únicas que aprovechen los datos altamente conectados de hoy.

# Apéndice A. Información y recursos adicionales

En esta sección, cubrimos rápidamente información adicional que puede ser útil para algunos lectores. Veremos otros tipos de algoritmos, otra forma de importar datos a Neo4j y otra biblioteca de procedimientos. También hay algunos recursos para encontrar conjuntos de datos, asistencia de plataforma y capacitación.

## Otros algoritmos

Se pueden usar muchos algoritmos con datos de gráficos. En este libro, nos hemos centrado en los que son más representativos de los algoritmos de gráficos clásicos y los que más utilizan los desarrolladores de aplicaciones. Algunos algoritmos, como la coloración y la heurística, se han omitido porque son más interesantes en los casos académicos o pueden derivarse fácilmente.

Otros algoritmos, como la detección de comunidades basadas en bordes, son interesantes, pero aún no se han implementado en Neo4j o Apache Spark. Esperamos que la lista de algoritmos de gráficos utilizados en ambas plataformas aumente a medida que crezca el uso de análisis de gráficos.

También hay categorías de algoritmos que se usan con gráficos pero que no son estrictamente grafológicos en su naturaleza. Por ejemplo, analizamos algunos algoritmos utilizados en el contexto del aprendizaje automático en el [Capítulo 8](#). Otra área de interés son los algoritmos de similitud, que a menudo se aplican a recomendaciones y predicción de enlaces. Los algoritmos de similitud determinan qué nodos se asemejan más entre sí utilizando varios métodos para comparar elementos como atributos de nodo.

## Neo4j importación de datos a granel y Yelp

Importando datos en Neo4j con el lenguaje de consulta encriptado utiliza un enfoque transaccional. [La Figura A-1](#) ilustra una descripción general de alto nivel de este proceso.

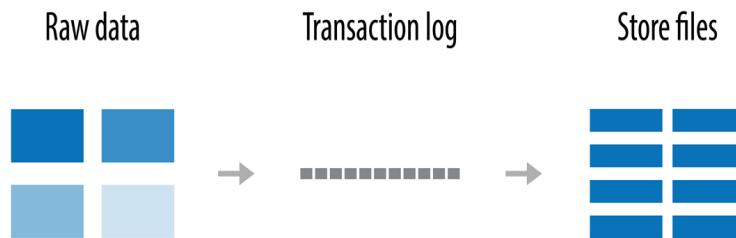


Figura A-1. Importación basada en cifrado

Si bien este método funciona bien para la carga incremental de datos o la carga masiva de hasta 10 millones de registros, la herramienta de importación Neo4j es una mejor opción cuando se importan conjuntos de datos masivos iniciales. Esta herramienta crea los archivos de la tienda directamente, omitiendo el registro de transacciones, como se muestra en la [Figura A-2](#).



Figura A-2. Usando la herramienta de importación Neo4j

La herramienta de importación Neo4j procesa archivos CSV y espera que estos archivos tengan encabezados específicos. [La Figura A-3](#) muestra un ejemplo de archivos CSV que pueden ser procesados por la herramienta.

Nodes	<code>id:ID(User)</code>	<code>name</code>	<code>id:ID(Review)</code>	<code>text</code>	<code>stars</code>
	1234	Bob	678	Awesome	3
1235	Alice	679	Mediocre	2	
1236	Erika	680	Really bad	1	

Relationships	<code>:START_ID(User)</code>	<code>:END_ID(Review)</code>
	1234	678
1235	679	
1236	680	

Figura A-3. Formato de archivos CSV que Neo4j Importa los procesos.

El tamaño del conjunto de datos de Yelp significa que la herramienta de importación Neo4j es la mejor opción para obtener los datos en Neo4j. Los datos están en formato JSON, por lo que primero debemos convertirlos al formato que espera la herramienta de importación Neo4j. [La Figura A-4](#) muestra un ejemplo del JSON que necesitamos transformar.

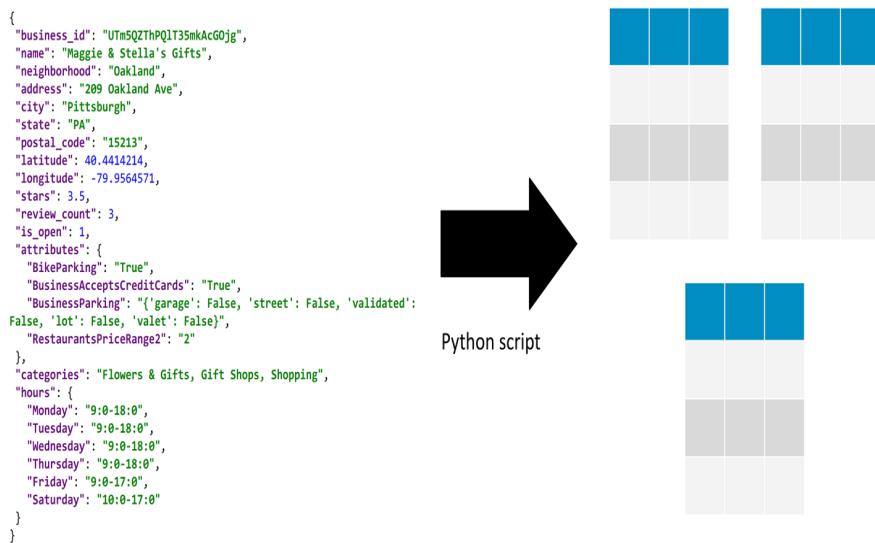


Figura A-4. Transformando JSON a CSV

Usando Python, podemos crear un script simple para convertir los datos a un archivo CSV. Una vez que hayamos transformado los datos en ese formato, podemos importarlos a Neo4j. Las instrucciones detalladas que explican cómo hacer esto se encuentran en el [repositorio de recursos del libro](#).

## APOC y otras herramientas Neo4j

[Awesome Procedures on Cypher \(APOC\)](#) es una biblioteca que contiene más de 450 procedimientos y funciones para ayudar con tareas comunes como la integración de datos, la limpieza de datos, la conversión de datos y las funciones de ayuda general. APOC es la biblioteca estándar para Neo4j.

Neo4j también tiene otras herramientas que se pueden usar en conjunto con su biblioteca de algoritmos de grafos, como una aplicación de "juegos" de algoritmos para la exploración sin código. Estos se pueden encontrar en su [sitio de desarrollador para algoritmos gráficos](#).

## Encontrar conjuntos de datos

Encontrar un conjunto de datos de grafía que se alinee con los objetivos de prueba o las hipótesis puede ser un desafío. Además de revisar los trabajos de investigación, considere la posibilidad de explorar índices para conjuntos de datos de red:

- [El Proyecto de análisis de red de Stanford \(SNAP\)](#) incluye varios conjuntos de datos junto con documentos relacionados y guías de uso.
- [El Índice de Redes Complejas de Colorado \(ICON, por sus siglas en inglés \)](#) es un índice de búsqueda de conjuntos de datos de redes con calidad de investigación de varios dominios de la ciencia de redes.
- [La Colección de la Red de Koblenz \(KONECT\)](#) incluye grandes conjuntos de datos de red de varios tipos para realizar investigaciones en la ciencia de la red.

La mayoría de los conjuntos de datos requerirán un poco de masaje para transformarlos en un formato más útil.

## Asistencia con las plataformas Apache Spark y Neo4j.

Hay muchos recursos en línea para las plataformas Apache Spark y Neo4j. Si tiene preguntas específicas, lo alentamos a que se comunique con sus respectivas comunidades:

- Para preguntas generales sobre Spark, suscríbase a users@spark.apache.org en [la página de Spark Community](#) .
- Para preguntas sobre GraphFrames, use el [rastreador de problemas de GitHub](#) .
- Para todas las preguntas de Neo4j (incluidos los algoritmos de gráficos), visite la [comunidad de Neo4j en línea](#) .

## Formación

Hay una serie de excelentes recursos para comenzar con el análisis gráfico. Una búsqueda de cursos o libros sobre algoritmos de gráficos, ciencia de redes y análisis de redes descubrirá muchas opciones. Algunos buenos ejemplos de aprendizaje en línea incluyen:

- [Curso de análisis de redes sociales aplicadas de Coursera en Python](#)
- [La serie de análisis de redes sociales de Leonid Zhukov en YouTube.](#)
- [El curso de Análisis de Redes de Stanford](#) incluye conferencias en video, listas de lectura y otros recursos
- [Complexity Explorer](#) ofrece cursos en línea en ciencias de la complejidad.

# Índice

---

## UNA

Algoritmo A \*

[Ruta más corta, Variación de la ruta más corta: A \\* - A \\* con Neo4j](#)

[con Neo4j, A \\* con Neo4j](#)

[densidad real, gráficos dispersos frente a gráficos densos](#)

[Gráficos acíclicos, Sabores de los Gráficos.](#)

[Gráficos cíclicos vs. Gráficos acíclicos versus Gráficos cíclicos - Árboles](#)

[árboles y, árboles](#)

[Mensajes agregados, ruta más corta \(ponderada\) con Apache Spark , ruta más corta de una sola fuente con Chispa Apache , centralidad de proximidad con Chispa Apache](#)

datos de vuelo de la aerolínea

[analizando con Apache Spark, analizando los datos de vuelo de las aerolíneas con Apache Spark - Aeropuertos interconectados por aerolínea](#)

[Retrasos de ORD, Retrasos de ORD - Retrasos de ORD](#)

análisis exploratorio, análisis [exploratorio](#)

retrasos relacionados con la niebla de la OFS, [Día malo en la OFS - Día malo en la OFS](#)

[Aeropuertos interconectados por aerolínea, Aeropuertos interconectados por aerolínea - Aeropuertos interconectados por aerolínea](#)

[Aeropuertos populares, Aeropuertos populares - Aeropuertos populares](#)

[Alexa, Gráficos, Contexto, y Precisión](#)

Procesamiento centrado en algoritmos, [consideraciones de procesamiento](#)

[Algoritmo de ruta más corta de todos los pares \(APSP\), Algoritmos de búsqueda de rutas y gráficos , Ruta más corta de todas las parejas - Ruta más corta de todas las parejas con Neo4j](#)

[secuencia de operaciones, una mirada más cercana a todos los pares camino más corto - una mirada más cercana a todos los pares camino más corto](#)

[¿Cuándo usar, cuándo debo usar la ruta más corta de todos los pares?](#)

[con Apache Spark, la ruta más corta para todos los pares con Apache Spark](#)

[con Neo4j, el camino más corto de todos los pares con Neo4j - el camino más corto de todos los pares con Neo4j](#)

[Amazon, Gráficos, Contexto y Precisión](#)

Análisis contra el lavado de dinero (AML), [Prólogo](#)

Chispa de apache

[sobre, Apache Spark - Instalando Spark](#)

[Algoritmo de ruta más corta de todos los pares con, Ruta más corta de todos los pares con Apache Spark](#)

[Análisis de datos de vuelos de aerolíneas con, Análisis de datos de vuelos de aerolíneas con Apache Spark - Aeropuertos interconectados por aerolínea](#)

(ver también datos de vuelo de la aerolínea)

[Amplia primera búsqueda con, Amplia primera búsqueda con Apache Spark](#)

[Centralidad de acercamiento con, Centralidad de acercamiento con Apache Spark - Centralidad de acercamiento con Apache Spark](#)

[Componentes conectados con, Componentes conectados con Apache Spark - Componentes conectados con Apache Spark](#)

[Grado de centralidad con, Grado de centralidad con Apache Spark](#)

[Importación de datos de gráficos sociales en, Importación de datos en Apache Spark](#)

[Importación de datos de gráficos de dependencia de software en, Importación de datos en Apache Spark](#)

[Importar el conjunto de datos de transporte en, Importar los datos en Apache Spark](#)

[instalar, instalar spark](#)

[Etiqueta de propagación con, Etiqueta de propagación con Apache Spark](#)

[Recursos en línea, asistencia con las plataformas Apache Spark y Neo4j.](#)

[PageRank con, PageRank con Apache Spark - PageRank hasta la convergencia](#)

[PageRank personalizado con PageRank personalizado con Apache Spark](#)

[Algoritmo de ruta más corta \(ponderada\), Ruta más corta \(ponderada\) con Apache Spark - Ruta más corta \(ponderada\) con Apache Spark](#)

[Algoritmo de ruta más corta de una sola fuente con, Ruta más corta de una sola fuente con Apache Spark - Ruta más corta de una sola fuente con Apache Spark](#)

Proyecto Spark Graph, [Apache Spark](#)

[Componentes fuertemente conectados con, Componentes fuertemente conectados con Apache Spark](#)

[Recuento de triángulos con, Recuento de triángulos con Apache Spark](#)

[Cuándo usarlo, seleccionando nuestra plataforma.](#)

Aproximación entre la centralidad, [Bellagio cruz-promoción](#)

[Inteligencia artificial \(IA\), aprendizaje automático y la importancia del contexto.](#)

grado medio, [alcance](#)

ruta media más corta, [Pathfinding](#)

[Procedimientos asombrosos en la biblioteca de Cypher \(APOC\), Neo4j Graph Platform , una descripción general rápida de los datos de Yelp , importación de datos en Neo4j, APOC y otras herramientas de Neo4j](#)

## segundo

[Número de tocino, ¿cuándo debo usar el camino más corto?](#)

[Barabási, Albert-László, ¿por qué debemos preocuparnos por los algoritmos de grafos?](#)

## [Centralidad intermediación de algoritmo](#), [Centralidad Algoritmos](#) , [intermediación Centralidad - Aproximación de centralidad de intermediación con Neo4j](#)

Puentes y puntos de control, [Puentes y puntos de control.](#)

[calculando, calculando la centralidad de intermediación](#)

[¿Cuándo usar, cuándo debo usar la centralidad de intermediación?](#)

[con Neo4j, centralidad de intermediación con Neo4j - Centralidad de intermediación con Neo4j](#)

[con el conjunto de datos de Yelp, Bellagio cross-promotion - Bellagio cross-promotion](#)

[Clasificación binaria, Cómo predecimos los enlaces faltantes](#)

[Gráficos bipartitos, Sabores de gráficos , Monopartitos, Bipartitos y k-Partite Graphs](#)

[Boruvka, Otakar, árbol de expansión mínima](#)

[Primera búsqueda de amplitud \(BFS\), Primera búsqueda de amplitud - Primera búsqueda de amplitud con Apache Spark](#)

[Puentes, puentes y puntos de control.](#)

[El problema de los puentes de Königsberg, ¿Qué son los gráficos? , Profundidad Primera Búsqueda](#)

[Importación de datos a granel, Herramienta de importación Neo4j para, Importación de datos a granel Neo4j y Yelp - Importación de datos a granel Neo4j y Yelp](#)

Paralelo síncrono a granel (BSP), [consideraciones de procesamiento](#)

## do

investigación del cáncer, [prólogo](#)

Algoritmos de centralidad, [centralidad , algoritmos de centralidad - Resumen](#)

[Centralidad de intermediación, Centralidad de intermediación - Aproximación de la centralidad de intermediación con Neo4j](#)

[Centralidad de proximidad, Centralidad de proximidad - Centralidad armónica con Neo4j](#)

[Grado de centralidad, Grado de centralidad - Grado de centralidad con Apache Spark](#)

[visión general, algoritmos de centralidad](#)

[Brandes aleatorios-aproximados, variación de centralidad de intermediación: Brandes aleatorios-aproximados](#)

[Datos de gráfico social para, Ejemplo de datos de gráfico: El gráfico social - Importando los datos en Neo4j](#)

Aeropuerto Internacional de Chicago O'Hare (ORD), datos en retrasos de, [retrasos de ORD - retrasos de ORD redes de citas, ¿cuándo debo usar los componentes conectados?](#)

[Clauset, A., ¿Qué son los análisis gráficos y los algoritmos?](#)

[camarilla, gráficos escasos frente a gráficos densos](#)

[La proximidad Centralidad algoritmo, Centralidad Algoritmos , la cercanía Centralidad - Centralidad armónica con Neo4j](#)

[Variación de centralidad armónica, variación de centralidad de proximidad: centralidad armónica](#)

[Variación de Wasserman Faust, variación de la proximidad de la centralidad: Wasserman y Fausto - Variación de la cercanía de la centralidad: Wasserman y Fausto](#)

[cuándo usar, cuándo debo usar la proximidad centralidad?](#)

[con Apache Spark, centralidad de proximidad con Apache Spark - Centralidad de proximidad con Apache Spark](#)

[con Neo4j, la proximidad centralidad con Neo4j](#)

Algoritmo de coeficiente de agrupamiento, [Algoritmos de detección de la comunidad](#)

(Ver también algoritmos de coeficiente de conteo y agrupación de triángulos)

[Clústeres, definidos, conectados frente a gráficos desconectados](#)

Índice de Colorado de redes complejas (ICON), [búsqueda de conjuntos de datos](#)

[algoritmos de detección de la comunidad, Detección de la comunidad , Algoritmos de detección de la comunidad - Validación de comunidades](#)

[Componentes conectados, Componentes conectados - Componentes conectados con Neo4j](#)

[para la predicción de enlaces, Predicción de enlaces: Detección comunitaria - Predicción de enlaces: Detección comunitaria](#)

[Propagación algoritmo de etiqueta, Propagación etiqueta - Propagación etiqueta con Neo4j](#)

[Modularidad de Louvain, Modularidad de Louvain - Louvain con Neo4j](#)

[Datos de gráfico de dependencia de software para, Ejemplo de datos de gráfico: El gráfico de dependencia de software - Importación de datos en Neo4j](#)

[Componentes fuertemente conectados, Componentes fuertemente conectados - Componentes fuertemente conectados con Neo4j](#)

[Recuento de triángulos y coeficiente de agrupamiento, Recuento de triángulos y coeficiente de agrupamiento - Coeficiente de agrupamiento local con Neo4j](#)

[Validando comunidades, Validando comunidades.](#)

[gráfico completo, gráficos dispersos frente a gráficos densos](#)

[componentes, definidos, Conectados Versus Gráficos Desconectados](#)

Configuración que supera a un solo hilo (COST), [Consideraciones de plataforma](#)

[Algoritmo de componentes conectados, algoritmos de detección de la comunidad , componentes conectados - Componentes conectados con Neo4j](#)

[¿Cuándo usar, cuándo debo usar los componentes conectados?](#)

[con Apache Spark, componentes conectados con Apache Spark - Componentes conectados con Apache Spark con Neo4j, componentes conectados con Neo4j](#)

[Gráficos conectados, Gráficos Conectados Versus Desconectados](#)

[contexto, prólogo , aprendizaje automático y la importancia del contexto](#)

[costos, ruta más corta , variación de la ruta más corta: A \\*](#)

[ciclos, gráficas acíclicas y gráficas cíclicas](#)

[Gráficos cíclicos, sabores de gráficos](#)

[Cypher, Apache Spark , Búsqueda de categorías similares , Importación de datos a granel Neo4j y Yelp](#)

[Cypher para Apache Spark \(CAPS\), Apache Spark](#)

**re**

[D'Orazio, Francesco, ¿por qué deberíamos preocuparnos por los algoritmos de grafos?](#)

[factor de amortiguación / amortiguación, iteración , usuarios aleatorios y sumideros de rango , PageRank con un número fijo de iteraciones](#)

[DataFrame, Apache Spark](#)

[Conjuntos de datos, fuentes para, Encontrar conjuntos de datos](#)

[deduplicación, ¿cuándo debo usar los componentes conectados?](#)

[Centralidad grado algoritmo, Centralidad Algoritmos , grado Centralidad - Centralidad grado con Spark Apache](#)

alcance de un nodo, [alcance](#)

[¿Cuándo usar, cuándo debo usar la centralidad del grado?](#)

[con datos del aeropuerto, Aeropuertos populares - Aeropuertos populares](#)

[con Apache Spark, grado centralidad con Apache Spark](#)

distribución de grado, [alcance](#)

grado de un nodo, [alcance](#)

[algoritmo de pasos delta, ruta más corta de una sola fuente con Neo4j](#)

[Gráficos densos, Sabores de gráficos , Gráficos dispersos y Gráficos densos](#)

[densidad de relaciones, algoritmos de detección de la comunidad](#)

[Primera búsqueda en profundidad \(DFS\), Primera búsqueda en profundidad - Primera búsqueda en profundidad , Componentes fuertemente conectados](#)

diámetro de una gráfica, [Pathfinding](#)

Dijkstra, Edsger, [camino más corto](#)

El algoritmo de Dijkstra (ver algoritmo de ruta más corta)

[Gráfico acíclico dirigido \(DAG\), Gráficos acíclicos versus Gráficos cílicos](#)

[Gráficos dirigidos, Sabores de los Gráficos , Gráficos no Dirigidos Versus Gráficos Dirigidos](#)

relaciones direccionales, [prólogo](#)

[Gráficos desconectados, Connected Versus Disconnected Graphs](#)

distancia (término), [camino más corto](#)

**mi**

[Eguílez, Víctor M., ¿Qué son los análisis gráficos y los algoritmos?](#)

Diagrama de relaciones entre entidades (ERD), [Prólogo](#)

[Erdős, Paul, ¿cuándo debo usar el camino más corto?](#)

Euler, Leonhard, [¿qué son los gráficos?](#)

[Camino de Euler, Primera búsqueda en profundidad](#)

**F**

[Facebook, Gráficos, Contexto y Precisión](#)

[Fausto, Katherine, variación de centralidad de proximidad: Wasserman y Fausto](#)

[Extracción de características, Extracción y selección de características conectadas](#)

[Importancia de la función, Predicción de enlaces: Características básicas del gráfico](#)

[Selección de características, Extracción y selección de características conectadas](#)

[Vectores de características, Extracción y Selección de Características Conectadas](#)

características

[conectado extracción de características / selección, conectado extracción de características y Selección - Características Gráfico algoritmo](#)

características del algoritmo gráfico, características del algoritmo [gráfico](#)

[Graphy, Graphy Features](#)

Fischer, Michael J., [Componentes conectados](#)

[Fleurquin, Pablo, ¿Qué son los análisis gráficos y los algoritmos?](#)

[foodweb, ¿Qué son los análisis gráficos y los algoritmos?](#)

[Freeman, Linton C., grado centralidad , centralidad intermedia](#)

**sol**

[Galler, Bernard A., Componentes conectados](#)

Punto de referencia Girvan – Newman (GN), [validación de comunidades](#)

Coeficiente de agrupamiento global, [Coeficiente de agrupamiento global](#)

[Patrones globales, procesamiento de gráficos, bases de datos, consultas y algoritmos.](#)

Google

PageRank, [PageRank](#)

[Pregel, Consideraciones de Procesamiento](#)

[Grandjean, Martin, ¿Qué son los análisis gráficos y los algoritmos?](#)

algoritmos gráficos (en general)

[sobre, ¿Qué son los análisis gráficos y los algoritmos? - ¿Qué son los análisis gráficos y los algoritmos?](#)

centralidad, [centralidad](#)

detección de la comunidad, detección de la [comunidad](#)

[Definidos, ¿Qué son los análisis gráficos y los algoritmos?](#)

(ver también algoritmos específicos)

[Importancia de, ¿Por qué debemos preocuparnos por los algoritmos de grafos? - ¿Por qué deberíamos preocuparnos por los algoritmos de grafos?](#)

[en la práctica, Graph Algorithms in Practice - Resumen](#)

búsqueda de caminos, [Pathfinding](#)

[tipos de, tipos de algoritmos de grafos](#)

analítica gráfica

[sobre, ¿Qué son los análisis gráficos y los algoritmos? - ¿Qué son los análisis gráficos y los algoritmos?](#)

[Definidos, ¿Qué son los análisis gráficos y los algoritmos?](#)

[casos de uso, Graph Analytics casos de uso](#)

[Motores de cálculo gráfico, Plataformas representativas.](#)

[Bases de datos gráficas, Plataformas representativas.](#)

[Incrustación de gráficos, definición, extracción de características conectadas y selección](#)

[Gráficos globales, procesamiento de gráficos, bases de datos, consultas y algoritmos , extracción y selección de características conectadas](#)

[Gráficos locales, procesamiento de gráficos, bases de datos, consultas y algoritmos , extracción de características conectadas y selección](#)

plataformas graficas

[Apache Spark, Apache Spark - Instalación de Spark](#)

[Neo4j, Neo4j Graph Platform - Instalación de Neo4j](#)

Consideraciones de la plataforma, [Consideraciones de la plataforma](#)

[Plataformas representativas, Plataformas representativas - Instalación de Neo4j](#)

[Seleccionando una plataforma, Seleccionando nuestra plataforma](#)

[Procesamiento de gráficos , procesamiento de gráficos, bases de datos, consultas y algoritmos - OLTP y OLAP , consideraciones de procesamiento](#)

[Algoritmos de búsqueda gráfica, búsqueda de caminos y algoritmos de búsqueda Gráfico - primera búsqueda en profundidad](#)

[Algoritmos de búsqueda de trazados y gráficos definidos](#)

[Gráficos de algoritmos de búsqueda, Pathfinding y algoritmos de búsqueda de gráficos.](#)

[Primera búsqueda de amplitud , Primera búsqueda de amplitud - Primera búsqueda de amplitud con Apache Spark](#)

[Primera búsqueda en profundidad , Primera búsqueda en profundidad - Primera búsqueda en profundidad](#)

[datos de gráficos de transporte para, Datos de ejemplo: El gráfico de transporte - Importando los datos a Neo4j](#)

[Teoría de la gráfica, Teoría de la gráfica y Conceptos - Resumen](#)

[sobre, Teoría de Gráficos y Conceptos - Resumen](#)

[Orígenes de, ¿Qué son las gráficas?](#)

terminología [terminología](#)

[tipos y estructuras, tipos de grafos y estructuras](#)

graficar algoritmos de recorrido

[Primera búsqueda de amplitud](#) , [Primera búsqueda de amplitud - Primera búsqueda de amplitud con Apache Spark](#)

[Primera búsqueda en profundidad](#) , [Primera búsqueda en profundidad - Primera búsqueda en profundidad](#)

Procesamiento centrado en el gráfico, [consideraciones de procesamiento](#)

[GraphFrames](#), [Apache Spark](#) , [PageRank con Apache Spark](#) , [importando los datos a Apache Spark](#)

gráficos (en general)

[sobre](#), [¿Qué son los gráficos?](#)

[Acyclic vs. Cyclic](#) , [Sabores de las gráficas](#) , [Graphic Acyclic Versus Cyclic Graphs - Trees](#)

[Bipartita](#) , [Sabores de las gráficas](#) , [Monopartita](#), [Bipartita y K-Partite Graphs](#)

atributos comunes, [sabores de grafos](#)

[Conectado frente a desconectado](#), [Conectado versus Gráficos desconectados](#)

[sabores de los sabores de los gráficos](#) : gráficos monopartitos, bipartitos y k-partitas

[Partituras k](#), [Sabores de las gráficas](#) , [Monopartitas](#), [Bipartitas y k Partitas](#)

[Monopartito](#), [sabores de gráficos](#) , [monopartitos](#), [bipartitos y k-Partite Graphs](#)

[escasos contra densos](#), [escasos gráficos frente a densos gráficos](#)

[no dirigido contra dirigido](#), [Sabores de gráficos](#) , [Gráficos no dirigidos versus Gráficos dirigidos](#)

[Sin ponderar vs. ponderado](#), [Sabores de los gráficos](#) , [Gráficos no ponderados y Gráficos ponderados](#)

conjuntos de datos de grafía, [búsqueda de conjuntos de datos](#)

graphy, características [Características grafía](#)

## H

[Camino hamiltoniano](#), [Primera búsqueda en profundidad](#)

[Algoritmo de proximidad de centralidad armónica](#), variación de centralidad de proximidad: centralidad armónica

[Hart](#), [Peter](#), [Variación del camino más corto: A \\*](#)

[Salto \(término\)](#), [Gráficos no ponderados y Gráficos ponderados](#) , [Ruta más corta](#)

Procesamiento híbrido transaccional y analítico (HTAP), [OLTP y OLAP](#)

## yo

[impureza](#), [Predicción de enlaces: Características básicas del gráfico](#)

[In-links](#), [Gráficos no direcccionados versus Gráficos dirigidos](#)

influencia, [influencia](#)

[islas](#), [conectadas versus gráficas desconectadas](#)

## K

[Gráficos k-partitas](#) , [Sabores de gráficos](#) , [Monopartitos](#), [Bipartitos y k-Partitas](#)

Colección de red de Koblenz (KONECT), [búsqueda de conjuntos de datos](#)

[El problema de los puentes de Königsberg](#), [¿qué son los gráficos?](#), [Profundidad Primera Búsqueda](#)

## L

[Algoritmo de propagación de etiquetas \(LPA\)](#), [Algoritmos de detección de la comunidad](#), [Propagación de etiquetas - Propagación de etiquetas con Neo4j](#)

[método de extracción](#), [propagación de etiquetas](#)

[método push](#), [propagación de etiquetas](#)

[Etiquetas de semillas](#), [Semi Supervisión de aprendizaje y Etiquetas de semillas](#)

[Aprendizaje semi supervisado](#), [Aprendizaje semi supervisado y etiquetas de semillas](#)

[¿Cuándo usar](#), [cuándo debo usar la propagación de etiquetas?](#)

[con Apache Spark](#), [propagación de etiquetas con Apache Spark](#)

[con Neo4j](#), [Etiqueta de propagación con Neo4j - Etiqueta de propagación con Neo4j](#)

[con el conjunto de datos de Yelp](#), [Buscar categorías similares - Encontrar categorías similares](#)

etiqueta, definido, [terminología](#)

Modelo de gráfico de propiedad etiquetada, [Terminología](#)

Punto de referencia de Lancichinetti – Fortunato – Radicchi (LFR), [Validación de comunidades](#)

[Puntos de referencia](#), [el camino más corto de todos los pares con Apache Spark](#)

[Latora, V.](#), [variación de centralidad de proximidad: centralidad armónica](#)

[Nodos foliares](#), [gráficos acíclicos y gráficos cíclicos](#).

[Lee, CY](#), [Breadth First Search](#)

[la predicción de enlaces](#), [gráficos y aprendizaje automático en la práctica: Enlace de predicción - Predicción de Enlaces: Detección Comunidad](#)

[Equilibrio / división de datos para entrenamiento / pruebas](#), [Equilibrio y división de datos - Equilibrio y división de datos](#)

[Características básicas del gráfico para](#), [Predicción de enlaces: Características básicas del gráfico - Predicción de enlaces: Características básicas del gráfico](#)

[Gráfico de coautor](#), [El gráfico de coautor](#)

[detección de la comunidad](#), [enlaces de predicción: Detección de la comunidad - Enlaces de predicción: detección de la comunidad](#)

[crear una formación equilibrada y probar conjuntos de datos](#), [crear una formación equilibrada y probar conjuntos de datos - equilibrar y dividir datos](#)

[Creación de canalización de aprendizaje automático](#), [Creación de un canal de aprendizaje automático definidas](#), [características del algoritmo grafico](#)

[importando datos a Neo4j](#), [importando los datos a Neo4j](#)

[Cómo predecir los enlaces que faltan](#), [Cómo predecimos los enlaces que faltan](#)

[Herramientas y datos](#), [Herramientas y datos - Herramientas y datos](#)

[Triángulos y coeficiente de agrupación, Predicción de enlaces: Triángulos y el coeficiente de agrupación - Predicción de enlaces: Triángulos y el coeficiente de agrupación](#)

descubrimiento basado en la literatura (LBD), [Prólogo](#)

[Coeficiente de agrupamiento local, Coeficiente de agrupamiento local , Coeficiente de agrupamiento local con Neo4j](#)

[La modularidad Lovaina algoritmo, algoritmos de detección de la Comunidad , Lovaina La modularidad - Lovaina con Neo4j](#)

[para la predicción de enlaces, Predicción de enlaces: Detección comunitaria - Predicción de enlaces: Detección comunitaria](#)

[agrupación basada en la calidad a través de la modularidad, agrupación basada en la calidad a través de la modularidad - agrupación basada en la calidad a través de la modularidad](#)

[¿Cuándo usar, cuándo debo usar Louvain?](#)

[con Neo4j, Lovaina con Neo4j - Lovaina con Neo4j](#)

## METRO

aprendizaje automático (ML)

[conectado extracción de características / selección, conectado extracción de características y Selección - Características Gráfico algoritmo](#)

[Incrustaciones de gráficos, Extracción y selección de características conectadas](#)

Gráficos, contexto y precisión, [Gráficos, Contexto y Precisión](#)

[Importancia del contexto en, El aprendizaje automático y la importancia del contexto.](#)

[Predicción de enlaces, uso de algoritmos gráficos para mejorar el aprendizaje automático](#)

[Marchiori, M., variación de centralidad de proximidad: centralidad armónica](#)

campañas de marketing, [prólogo](#)

[matplotlib, una descripción rápida de los datos de Yelp](#)

[densidad máxima, gráficos dispersos frente a gráficos densos](#)

[Expansión mínima algoritmo de árbol, Pathfinding y Gráfico algoritmos de búsqueda , árbol de expansión mínima - árbol de expansión mínimo con Neo4j](#)

[cuándo usar, cuándo debo usar el árbol de expansión mínima?](#)

[con Neo4j, árbol de expansión mínimo con Neo4j](#)

[Modularidad, agrupación basada en la calidad vía modularidad.](#)

(Véase también el algoritmo de modularidad de Lovaina)

[cálculo, agrupación basada en la calidad mediante modularidad - agrupación basada en la calidad mediante modularidad](#)

[agrupación basada en la calidad y agrupación basada en la calidad mediante modularidad - agrupación basada en la calidad mediante modularidad](#)

lavado de dinero, [prólogo](#)

[Gráficos monopartidos, Sabores de gráficos , Monopartidos, Bipartidos y k-Partite Graphs](#)

[Moore, C., ¿Qué son los análisis gráficos y los algoritmos?](#)[Moore, Edward F., Breadth First Search](#)[Multigraph, tipos de gráficos y estructuras](#)**norte**[pesos negativos, ¿cuándo debo usar el camino más corto?](#)

Neo4j

[Algoritmo A \\* con, A \\* con Neo4j](#)[Algoritmo de ruta más corta de todos los pares con, Ruta más corta de todos los pares con Neo4j - Ruta más corta de todos los pares con Neo4j](#)[analizar datos de Yelp con, analizar datos de Yelp con Neo4j - Encontrar categorías similares](#)

(ver también el conjunto de datos de Yelp)

[Centralidad de intermediación con, Centralidad de intermediación con Neo4j - Centralidad de intermediación con Neo4j](#)[Centralidad de proximidad con, Centralidad de proximidad con Neo4j](#)[Componentes conectados con, Componentes conectados con Neo4j](#)importando el conjunto de datos de la red de citas en, importando [los datos a Neo4j](#)[Importación de datos de gráficos sociales en, \[Importación de datos en Neo4j\]\(#\)](#)[Importación de datos de gráficos de dependencia de software en, \[Importación de datos en Neo4j\]\(#\)](#)Importar el conjunto de datos de transporte en, [Importar los datos en Neo4j](#)[Etiqueta de propagación con, Etiqueta de propagación con Neo4j - Etiqueta de propagación con Neo4j](#)[Coeficiente de agrupamiento local con, Coeficiente de agrupamiento local con Neo4j](#)[Modularidad de Louvain con, Louvain con Neo4j - Louvain con Neo4j](#)Algoritmo de árbol de expansión mínimo con, [Árbol de expansión mínimo con Neo4j](#)[Recursos en línea, asistencia con las plataformas Apache Spark y Neo4j.](#)[PageRank con, PageRank con Neo4j - PageRank con Neo4j](#)Algoritmo de paseo aleatorio con, [Paseo aleatorio con Neo4j](#)[Brandes aleatorios-aproximados con, aproximación de centralidad de intermediación con Neo4j](#)[Algoritmo de ruta más corta \(no ponderado\), Ruta más corta con Neo4j - Ruta más corta con Neo4j](#)Algoritmo de ruta más corto (ponderado), [Ruta más corta \(ponderado\) con Neo4j](#)Algoritmo de ruta más corta de una sola fuente con, [Ruta más corta de una sola fuente con Neo4j](#)[Componentes fuertemente conectados con, Componentes fuertemente conectados con Neo4j - Componentes fuertemente conectados con Neo4j](#)[Triángulos con, Triángulos con Neo4j](#)[Cuándo usarlo, seleccionando nuestra plataforma.](#)[Algoritmo de las rutas más cortas k de Yen, Variación de la ruta más corta: Las rutas k más cortas de Yen](#)

## Biblioteca de algoritmos Neo4j

[Ruta más corta \(sin ponderar\)](#), [Ruta más corta con Neo4j - Ruta más corta con Neo4j](#)

[Ruta más corta \(ponderada\)](#), [Ruta más corta \(ponderada\) con Neo4j](#)

[Neo4j Desktop](#), [Instalación de Neo4j](#)

[Neo4j Graph Platform](#), [Neo4j Graph Platform - Instalación de Neo4j](#)

[Herramienta de importación Neo4j](#), [Importación de datos a granel Neo4j y Yelp - Importación de datos a granel Neo4j y Yelp](#)

[¿Qué son los análisis gráficos y los algoritmos de redes ?](#)

redes

La gráfica como representación de [¿Qué son las gráficas?](#)

[tipos y estructuras](#), [tipos de grafos y estructuras](#)

[Newman, MEJ](#), [¿Qué son los análisis gráficos y los algoritmos?](#)

[Nilsson, Nils](#), [Variación del camino más corto: A \\*](#)

[Procesamiento centrado en nodos](#), [Consideraciones de procesamiento](#)

nodos

[Centralidad y, algoritmos de centralidad.](#)

[definido](#), [¿qué son las gráficas?](#)

## O

Procesamiento analítico en línea (OLAP), [OLTP y OLAP](#)

aprendizaje en línea, [entrenamiento](#)

Procesamiento de transacciones en línea (OLTP), [OLTP y OLAP](#)

[Out-links](#), [Gráficos no Dirigidos Versus Dirigidos](#)

## PAG

Page, Larry, [PageRank](#)

[PageRank](#), [algoritmos de centralidad](#), [PageRank - PageRank personalizado con Apache Spark](#)

e influencia, [influencia](#)

Implementación de la convergencia, [PageRank hasta la convergencia.](#)

[fórmula para](#), [la fórmula de PageRank](#)

[iteración / surfistas aleatorios / rangos de rango](#), [iteración](#), [surfistas aleatorios y fregaderos de rango](#)

[Variante de PageRank personalizado](#), [Variación de PageRank: PageRank personalizado](#)

[¿Cuándo usar](#), [cuándo debo usar PageRank?](#)

[con Apache Spark](#), [PageRank con Apache Spark - PageRank hasta la convergencia](#)

[con un número fijo de iteraciones](#), [PageRank con un número fijo de iteraciones](#)

[con Neo4j, PageRank con Neo4j - PageRank con Neo4j](#)

[con el conjunto de datos de Yelp, Búsqueda de críticos de hoteles influyentes - Búsqueda de críticos de hoteles influyentes](#)

[biblioteca de pandas, una descripción rápida de los datos de Yelp](#)

[Distribución de Pareto, ¿Por qué deberíamos preocuparnos por los algoritmos de grafos?](#)

camino, definido, [terminología](#)

[Los algoritmos de búsqueda de caminos, Pathfinding , Pathfinding y el gráfico de algoritmos de búsqueda - importar los datos en Neo4j](#)

[La ruta más corta de todos los pares, la ruta más corta de todos los pares - La ruta más corta de todos los pares con Neo4j](#)

[Algoritmo de árbol de expansión mínimo, Árbol de expansión mínimo - Árbol de expansión mínimo con Neo4j](#)

[Algoritmo de paseo aleatorio, Paseo aleatorio - Paseo aleatorio con Neo4j](#)

[Ruta más corta, Ruta más corta - Yen con Neo4j](#)

[La ruta más corta de una sola fuente, la ruta más corta de una sola fuente - La ruta más corta de una sola fuente con Neo4j](#)

[datos de gráficos de transporte para, Datos de ejemplo: El gráfico de transporte - Importando los datos a Neo4j](#)

[Gráficos ponderados y, Gráficos no ponderados versus Gráficos ponderados](#)

Pearson, Karl, [Random Walk](#)

[PageRank personalizado \(PPR\), ¿cuándo debo usar PageRank? , Variación del PageRank: PageRank personalizado](#)

[Nodos pivotales, puentes y puntos de control.](#)

[ley de poder, ¿Por qué nos deberían importar los algoritmos de grafos?](#)

[Acoplamiento preferencial, ¿por qué deberíamos preocuparnos por los algoritmos de grafos?](#)

[Pregel, Consideraciones de Procesamiento](#)

[Algoritmo de Prim, árbol de expansión mínima](#)

Motores de recomendación de productos, [gráficos, contexto y precisión.](#)

propiedades definidas, [terminología](#)

[Pseudografo, Tipos y Estructuras de Graficos.](#)

[pyspark REPL, Instalando Spark](#)

## Q

[agrupación basada en la calidad , agrupación basada en la calidad a través de la modularidad - agrupación basada en la calidad a través de la modularidad](#)

## R

Raghavan, Usha Nandini, [propagación de etiquetas](#)

[Ramasco, José J., ¿Qué son los análisis gráficos y los algoritmos?](#)

[bosque aleatorio, Cómo predecimos los enlaces faltantes](#), [Predicción de enlaces: Características básicas de Graph](#)

[Red aleatoria, aleatorio, pequeño mundo, estructuras sin escala](#)

[Algoritmo de recorrido aleatorio, Pathfinding y Gráfico algoritmos de búsqueda](#), [Random Walk - paseo aleatorio con Neo4j](#)

[¿Cuándo usar, cuándo debo usar la caminata aleatoria?](#)

[con Neo4j, Random Walk con Neo4j](#)

Algoritmo de centralidad de Brandes aproximadas aleatorias (RA-Brandes), [Variación de centralidad de la intermediación: Brandes aproximados aleatorios](#)

[Fregadero de rango, Iteración, Surfistas aleatorios y Fregaderos de rango](#)

[Raphael, Bertram, Variación del camino más corto: A \\*](#)

alcance de un nodo, [alcance](#)

[Reif, Jennifer, instalando Neo4j](#)

tipo de relacion, [terminología](#)

Procesamiento centrado en la relación, [consideraciones de procesamiento](#)

relaciones (término), [Introducción](#), [¿Qué son las gráficas?](#)

## S

Aeropuerto Internacional de San Francisco (SFO), datos en retrasos relacionados con la niebla de [Bad Day at SFO - Bad Day at SFO](#)

[Red sin escala, aleatorio, pequeño mundo, estructuras sin escala](#)

[ley de escala \(ley de poder\)](#), [¿Por qué nos deberían importar los algoritmos de grafos?](#)

buscadores, [prólogo](#)

[Etiquetas de semillas, Semi Supervisión de aprendizaje y Etiquetas de semillas](#)

[Aprendizaje semi supervisado, Aprendizaje semi supervisado y etiquetas de semillas](#)

Siete puentes de Königsberg problema, [¿Qué son los gráficos?](#), [Profundidad Primera Búsqueda](#)

[Algoritmo de ruta más corta, algoritmos de búsqueda de rutas y gráficos](#), [ruta más corta - Yen con Neo4j](#)

[Algoritmo A \\*, Variación de la ruta más corta: A \\* - A \\* con Neo4j](#)

[¿Cuándo usar, cuándo debo usar el camino más corto?](#)

[con Apache Spark \(ponderada\)](#), [ruta más corta \(ponderada\) con Apache Spark - Ruta más corta \(ponderada\)](#), [con Apache Spark](#)

[con Neo4j \(no ponderado\)](#), [la ruta más corta con Neo4j - la ruta más corta con Neo4j](#)

[con Neo4j \(ponderado\)](#), [el camino más corto \(ponderado\) con Neo4j](#)

[Variación de k-rutas más cortas de yenes](#), [la ruta más corta Variación: k-cortos de yenes Caminos](#)

[Gráficos simples, tipos de gráficos y estructuras.](#)

[Algoritmo de ruta más corta de una sola fuente \(SSSP\)](#), [Algoritmos de búsqueda de rutas y gráficos](#) , [Ruta más corta de una sola fuente - Ruta más corta de una sola fuente con Neo4j](#)

[con Apache Spark](#), [la ruta más corta de una sola fuente con Apache Spark - La ruta más corta de una sola fuente con Apache Spark](#)

[con Neo4j](#), [la ruta más corta de una sola fuente con Neo4j](#)

[Red de mundo pequeño](#), [Estructuras aleatorias](#), [Mundo pequeño](#), [sin escala](#).

datos de la gráfica social

[para algoritmos de centralidad](#), [ejemplo de datos de gráfico: el gráfico social - Importando los datos a Neo4j](#)  
[importando a Apache Spark](#), [importando los datos a Apache Spark](#)

[importando a Neo4j](#), [importando los datos a Neo4j](#)

[análisis de redes sociales](#), [¿cuándo debo usar el recuento de triángulos y el coeficiente de agrupación?](#)

[datos de gráficos de dependencia de software](#), [ejemplo de datos de gráficos: el gráfico de dependencia de software - Importando los datos a Neo4j](#)

[importando a Apache Spark](#), [importando los datos a Apache Spark](#)

[importando a Neo4j](#), [importando los datos a Neo4j](#)

árboles de expansión, [árboles](#)

Proyecto Spark Graph, [Apache Spark](#)

[gráficos dispersos](#), [sabores de gráficos](#) , [gráficos dispersos frente a gráficos densos](#)

Proyecto de análisis de red de Stanford (SNAP), [búsqueda de conjuntos de datos](#)

[Gráficos estrictos](#), [tipos de gráficos y estructuras](#).

[Strogatz, Steven](#), [¿por qué deberíamos preocuparnos por los algoritmos de grafos?](#)

[Algoritmo de componentes fuertemente conectados \(SCC\)](#), [algoritmos de detección de la comunidad](#) , [componentes fuertemente conectados - Componentes fuertemente conectados con Neo4j](#)

[¿Cuándo usar](#), [cuándo debo usar componentes fuertemente conectados?](#)

[con datos de aeropuertos](#), [aeropuertos interconectados por aerolínea - aeropuertos interconectados por aerolínea](#)

[con Apache Spark](#), [componentes fuertemente conectados con Apache Spark](#)

[con Neo4j](#), [componentes fuertemente conectados con Neo4j - Componentes fuertemente conectados con Neo4j](#)

[Agujero estructural](#), [coeficiente de agrupación local con Neo4j](#)

subgrafo, definido, [terminología](#)

## T

[Teletransportación](#), [iteración](#), [surfistas aleatorios y sumideros de rango](#)

[Pruebas de conjuntos de datos](#), [Creación de formación equilibrada y Pruebas de datos : equilibrio y división de datos](#)

## [conjuntos de datos de formación, Creación de conjuntos de datos de formación y pruebas equilibrados - Equilibrio y división de datos](#)

formación, recursos en línea para, [formación](#)

relaciones transitivas, [prólogo](#)

## [Translytics, OLTP y OLAP](#)

[Conjuntos de datos de transporte, Datos de ejemplo: el gráfico de transporte - Importando los datos a Neo4j](#)

[importando a Apache Spark, importando los datos a Apache Spark](#)

[importando a Neo4j, importando los datos a Neo4j](#)

Problema de vendedor ambulante (TSP), [búsqueda en profundidad primero](#)

Procesamiento transversal-transversal, [Consideraciones de procesamiento](#)

árboles [árboles](#)

Trémaux, Charles Pierre, [Primera búsqueda en profundidad](#)

[Conde triángulo y algoritmos coeficiente de agrupamiento, Detección Algoritmos Comunidad , triángulo Count and Clustering Coeficiente - coeficiente de la agrupación local con Neo4j](#)

[para la predicción de enlaces \(ejemplo de aprendizaje automático\), enlaces de predicción: triángulos y el coeficiente de agrupamiento - Enlaces de predicción: triángulos y el coeficiente de agrupamiento](#)

Coeficiente de agrupamiento global, [Coeficiente de agrupamiento global](#)

coeficiente de agrupación local, coeficiente de agrupación [local](#)

coeficiente de agrupamiento local con Neo4j, [coeficiente de agrupamiento local con Neo4j](#)

Recuento de triángulos con chispa de apache, [conteo de triángulos con chispa de apache](#)

Triángulos con Neo4j, [Triángulos con Neo4j](#)

[cuándo usar, cuándo debo usar el recuento de triángulos y el coeficiente de agrupamiento?](#)

[viaje de aplicación de planificación, de viaje Planificación App - Consulting viajes de negocios](#)

Gorjeo

[Propagación de etiquetas, ¿cuándo debo usar la propagación de etiquetas?](#)

[PageRank personalizado, ¿cuándo debo usar PageRank?](#)

## **U**

[Gráficos no dirigidos, Sabores de gráficos , Gráficos no direccionaldos y Gráficos dirigidos](#)

[Union Find, Componentes Conectados](#)

[Gráficos no ponderados, Sabores de gráficos , Gráficos no ponderados y Gráficos ponderados](#)

[las rutas más cortas no ponderadas, la ruta más corta con Neo4j - la ruta más corta con Neo4j](#)

## **V**

[vértices, ¿qué son las gráficas?](#)

(ver también nodos)

**W**

[Wasserman Faust algoritmo de proximidad, variación de centralidad de proximidad: Wasserman y Fausto - Variación de centralidad de proximidad: Wasserman y Fausto](#)

[Wasserman, Stanley, variación de centralidad de proximidad: Wasserman y Fausto](#)

Componentes débilmente conectados, [Componentes conectados](#)

peso (término), [camino más corto](#)

[Gráficos ponderados, Sabores de gráficos , Gráficos no ponderados y Gráficos ponderados](#)

Caminos más cortos ponderados

[con Apache Spark, ruta más corta \(ponderada\) con Apache Spark - Ruta más corta \(ponderada\) con Apache Spark](#)

[con Neo4j, el camino más corto \(ponderado\) con Neo4j](#)

[propiedad de peso, PageRank con Neo4j](#)

**Y**

Conjunto de datos de Yelp

[analizar con Neo4j, analizar datos de Yelp con Neo4j - Encontrar categorías similares](#)

[Bellagio promoción cruzada, Bellagio promoción cruzada - Bellagio promoción cruzada](#)

[Encontrar críticos de hoteles influyentes, Encontrar críticos de hoteles influyentes - Travel Business Consulting](#)

[Encontrar categorías similares, Encontrar categorías similares - Encontrar categorías similares](#)

modelo gráfico, modelo [gráfico](#)

[Importando en Neo4j, Importación de datos](#)

[Herramienta de importación Neo4j para, Importación de datos a granel Neo4j y Yelp - Importación de datos a granel Neo4j y Yelp](#)

[descripción general, una descripción general rápida de los datos de Yelp - una descripción general rápida de los datos de Yelp](#)

[red social, red social de Yelp](#)

[Consultoría de viajes , Consultoría de negocios de viajes - Consultoría de negocios de viajes](#)

[viaje de aplicación de planificación, de viaje Planificación App - Consulting viajes de negocios](#)

[Algoritmo de las rutas más cortas k de Yen, Variación de la ruta más corta: Las rutas k más cortas de Yen](#)

[Yen, Jin Y., Variación de la ruta más corta: k de Yen, las rutas más cortas](#)

## Sobre los autores

Mark Needham es defensor de gráficos e ingeniero de relaciones con desarrolladores en Neo4j. Trabaja para ayudar a los usuarios a adoptar gráficos y Neo4j, creando soluciones sofisticadas para problemas de datos desafiantes. Mark tiene una amplia experiencia en datos gráficos, habiendo ayudado anteriormente a construir el sistema de agrupamiento causal de Neo4j. Escribe sobre sus experiencias como grafista en su popular blog en <https://markhneedham.com/blog/> y tweets [@markhneedham](#) .

Amy E. Hodler es una devota de ciencia de redes y gerente de programas de analítica gráfica y inteligencia artificial en Neo4j. Promueve el uso de análisis de gráficos para revelar estructuras dentro de redes del mundo real y predecir el comportamiento dinámico. Amy ayuda a los equipos a aplicar enfoques novedosos para generar nuevas oportunidades en empresas como EDS, Microsoft, Hewlett-Packard (HP), Hitachi IoT y Cray Inc. Amy ama la ciencia y el arte con una fascinación por los estudios de complejidad y la teoría de grafos. Ella tweets [@amyhodler](#) .

## Colofón

El animal en la portada de Graph Algorithms es la araña de jardín europea ( *Araneus diadematus* ), una araña común de Europa y también de América del Norte, donde fue introducida inadvertidamente por los colonos europeos.

La araña europea de jardín mide menos de una pulgada de largo y está manchada de color marrón con marcas pálidas, algunas de las cuales en su parte posterior están dispuestas de tal manera que parecen formar una pequeña cruz, dando a la araña su nombre común de "araña cruzada". Estas arañas son comunes en su rango y se notan más a menudo a fines del verano, a medida que crecen a su tamaño más grande y comienzan a tejer sus redes.

Las arañas de jardín europeas son tejedoras de orbes, lo que significa que tejen una red circular en la que atrapan a sus pequeñas presas de insectos. La red a menudo se consume y respeta por la noche para garantizar y mantener su eficacia. Mientras la araña permanece fuera de la vista, una de sus patas descansa sobre una "línea de señal" conectada a la red, movimiento que alerta a la araña de la presencia de presas en apuros. La araña luego se mueve rápidamente para morder a su presa para matarla y también inyectarla con enzimas especiales que permiten el consumo. Cuando sus redes son perturbadas por depredadores o por disturbios involuntarios, las arañas de jardín europeas usan sus piernas para sacudir su red, y luego caen al suelo con un hilo de su seda. Cuando pasa el peligro, la araña usa este hilo para volver a subir a su web.

Viven durante un año: después de la incubación en primavera, las arañas maduran durante el verano y se aparean a finales de año. Los machos se acercan a las hembras con precaución, ya que las hembras a veces matan y consumen a los machos. Después de aparearse, la araña hembra teje un denso capullo de seda para sus huevos antes de morir en el otoño.

Al ser bastante comunes y adaptarse bien a los hábitats perturbados por el hombre, estas arañas están bien estudiadas. En 1973, dos arañas hembras, llamadas Arabella y Anita, formaron parte de un experimento a bordo del orbitador Skylab de la NASA, para probar el efecto de la gravedad cero en la construcción de la red. Después de un período inicial de adaptación al entorno sin peso, Arabella construyó una red parcial y luego una red circular completamente formada.

Muchos de los animales en las cubiertas de O'Reilly están en peligro de extinción; Todos ellos son importantes para el mundo.

La imagen de la portada es una ilustración en color de Karen Montgomery, basada en un grabado en blanco y negro de Meyers Kleines Lexicon . Las fuentes de portada son Gilroy y Guardian Sans. La fuente del texto es Adobe Minion Pro; la fuente del encabezado es Adobe Myriad Condensed; y la fuente del código es Ubuntu Mono de Dalton Maag.