Universidad ORT Uruguay

Facultad de Ingeniería

Uso de Neo4J como base de datos orientada a grafos para la gestión de información de paradas, recorridos y viajes del Sistema de Transporte Metropolitano (STM) de la Ciudad de Montevideo, Uruguay

Entregado como requisito para la obtención del título del Máster en Big Data

José Enrique Peirano - 252190

Juan Camilo Rojas – 235895

Augusto Romero - 253261

Tutores:

Juan Gabito

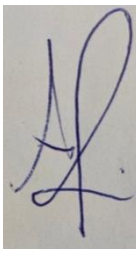
Sergio Yovine

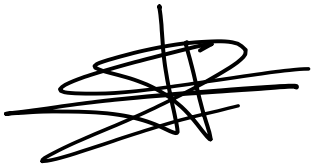
2021

Declaración de autoría

Nosotros, Augusto Romero, José Enrique Peirano y Juan Camilo Rojas, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Master en Big Data;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.





_____	_____	_____
Augusto Romero	*Juan Camilo Rojas*	*José Enrique Peirano*
24/09/2021	*24/09/2021*	*24/09/2021*

Agradecimientos

Al Departamento de Movilidad de la Intendencia de Montevideo y la Dirección de Análisis de Datos quienes nos permitieron complementar la información publicada en el portal de la Agencia de Gobierno Electrónico y Sociedad de la Información y del Conocimiento, quienes además nos compartieron su caso, sus actuales proyectos y sus necesidades en torno a la gestión de la información del Sistema de Transporte Metropolitano en Montevideo, desde donde parte el desarrollo de esta tesis y a quienes esperamos que el resultado aquí expuesto les genere valor agregado y les aporte para sus actuales o futuros proyectos.

A todo el equipo docente y administrativo del Máster en Big Data en la Facultad de Ingeniería de la Universidad ORT quienes, durante el tiempo de cursada, sin pandemia y con pandemia de por medio, no dudaron en prestarnos su mejor servicio e involucrarnos en un excelente proceso de aprendizaje entorno al Big Data. A Sergio Yovine quien no solo fue tutor para esta tesis sino quien fue nuestra guía principal durante toda la cursada del máster. Y al profesor Juan Gabito, quien con su excelente conocimiento en bases de datos avanzadas nos acompañó como tutor y nos aportó una retroalimentación constante en el desarrollo académico de esta tesis.

A nuestra familia, parejas y amigos quienes nos alentaron y apoyaron para terminar este proceso dando lo mejor.

Abstract

La utilización de bases de datos de grafos muestra un crecimiento muy importante en los últimos anos, potenciando el manejo de grandes volúmenes de datos, por lo que acompaña perfectamente el desarrollo actual del Big Data.

Resulta de especial interés para este trabajo la utilización de Neo4j para explotar los datos de movilidad de pasajeros del Sistema de Transporte Metropolitano (STM) de Montevideo, Uruguay.

Se analizaron otros estudios disponibles sobre movilidad y transporte con bases de datos orientadas a grafos, donde destaca particularmente el concepto de las *Time Varying Graphs* (TVG) como forma de modelar los datos para su posterior explotación, y también la utilización de Graph Neural Networks (GNN) para utilizar como mecanismo de predicción de la demanda de transporte.

Este trabajo realiza una prueba de concepto sobre los datos obtenidos de las tarjetas STM de un día en particular, testeando algunas de las variantes de explotación de datos y consultas en el browser de *Neo4j* así como la aplicación de algoritmos de aprendizaje automático como *shortest-path, pagerank o k-means*, trabajando con datos temporales, espaciales y espacio-temporales. Por último, se describe a nivel teórico la relación existente entre Neo4j y el ecosistema de Apache *Spark*, que permite que *Neoj4* tenga acceso a las fuentes de datos que ya están conectadas a *Spark*.

Se concluye que en el análisis de tránsito y movilidad es importante la utilización de bases de datos orientadas a grafos por la versatilidad de su modelado y la posibilidad de capturar grandes volúmenes de datos sobre los que se puedan realizar consultas refinadas, recorriendo los grafos con consultas en *Cypher* o aplicando algoritmos que provean de insumos relevantes para la toma de decisiones.

Finalmente, a la luz de los resultados obtenidos y del alcance del trabajo realizado, se realizan una serie de recomendaciones, en cuanto a la integración con *Spark* y con capas de visualización, para que la Intendencia de Montevideo o el ente usuario de los datos, pueda refinar los resultados e implementarlos.

Palabras claves

Grafo; Nodo; Relación; Propiedad; Parada; Recorrido; Viaje; Línea; Variante de línea; Neoj4; Cypher; STM; Time Varying Graphs; Spark

INDICE

Tabla de Ilustraciones

1. Objetivos

1.1. Objetivo general

Profundizar nuestros conocimientos sobre bases de datos orientadas a grafos dentro del ecosistema de Neo4j, buscando realizar una prueba de concepto en base a un *dataset* público con datos de Uruguay, documentando los hallazgos.

Utilizar la información disponible y suministrada por la Intendencia de Montevideo relacionada con el Sistema de Transporte Metropolitano (de aquí en más STM) para extraer conocimiento y responder preguntas que impacten en la gestión de rutas, viajes y paradas utilizando Neo4j como base de datos orientada a grafos para su desarrollo.

1.2. Objetivos específicos

1. Conocer y analizar los conceptos teóricos en torno a las bases de datos orientadas a grafos (ejemplos, modelos, lenguaje de consulta, algoritmos que se puedan aplicar, entre otros).
2. Investigar casos de aplicación de las bases de datos orientadas a grafos en sistemas de movilidad o transporte que puedan servir como base o estado del arte de este trabajo final.
3. Investigar y entender potenciales necesidades que tenga la Intendencia de Montevideo en cuanto a la gestión del Sistema de Transporte Metropolitano (STM) que puedan ser resueltas mediante la analítica de datos almacenados en una base de datos orientada a grafos.
4. Modelar, extraer, transformar y cargar los datos del STM en Neo4j para tenerlos disponibles en su motor de grafos.
5. Explorar los datos importados dentro de la herramienta para conocer las funcionalidades y obtener algunas respuestas básicas y recurrentes usando funcionalidades nativas y consultas con Cypher.
6. Responder consultas más avanzadas utilizando algoritmos disponibles en Neo4j para identificar potenciales oportunidades para la explotación de la información dentro de la herramienta.

7. Enmarcar el trabajo realizado con Neo4J dentro de un modelo de arquitectura viable para el caso de estudio analizado.

2. Introducción a grafos

A continuación, algunos conceptos principales sobre grafos que fueron tenidos en cuenta a la hora de desarrollar este trabajo final.

2.1. Teoría de grafos

La teoría de grafos es una rama de las matemáticas y ciencias de la computación que estudia las propiedades de los grafos.

Formalmente un grafo, representado como G(V,E), es un una pareja ordenada de V y E, en la que V es un conjunto no vacío de vértices y E es un conjunto de aristas que consta de pares no ordenados de vértices tales como {x,y}. X e y son adyacentes y se representan mediante una línea no orientada que une dichos vértices [1].

La teoría de grafos tiene sus fundamentos en las matemáticas discretas y en las aplicadas. Esta teoría requiere de diferentes conceptos de áreas como: combinatoria, álgebra, probabilidad, geometría de polígonos, aritmética y topología [2]

2.2. Características de un grafo

En la teoría, un grafo, usando la misma terminología que usa Neo4j, que es un Software de bases de datos orientados a grafos ampliamente utilizado y de código abierto, está compuesto por [3]:

- Relaciones: Gráficamente corresponden a las aristas o líneas que unen los vértices. Para Neo4j esta característica genera una relación entre los puntos que conecta.
- Nodos: Son los vértices que unen las relaciones.
- Camino: conjunto de vértices interconectados por aristas
- Propiedades: son características adicionales que se le pueden asignar tanto a los nodos como a las relaciones para otorgar información adicional y enriquecer el modelo.

Adicionalmente existen distintos tipos de grafos:

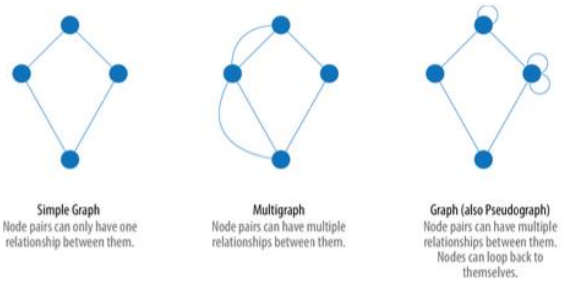


Ilustración 1 Tipos de grafos [3]

- Simple: Acepta una sola relación entre dos nodos.
- Multigrafo: Acepta múltiples relaciones entre dos nodos
- Pseudografo: Acepta múltiples relaciones entre nodos, incluyendo relaciones entre sí mismos.

2.3. Grafos y Big Data

Actualmente, la teoría de grafos está teniendo mayor influencia en el campo de las tecnologías de la información debido a la gran cantidad de aplicaciones en la optimización de recorridos, procesos, flujos y algoritmos de búsqueda, entre otros. Por ejemplo, en el año 1999 Google comenzó la tendencia del análisis de grafos en la era moderna utilizando vínculos entre documentos en la web para entender el contexto semántico, generando así un motor de búsqueda que superó ampliamente a sus competidores [4]. A raíz de ello el enfoque centrado en grafos ha generado innovación no solo en su mercado de búsquedas sino en todo el espacio de gestión de información.

No solo Google con sus recursos virtualmente ilimitados ha sabido aprovechar esta tecnología. Neo4j es una base de datos basada en grafos madura y de código abierto, que está siendo usada en producción en todo tipo de organizaciones, desde corporaciones globales como Walmart, Lufthansa y Cisco, a startups innovadoras como Fifty Three, Medium y CrunchBase [5]

Las bases de datos orientadas a grafos han cobrado notoriedad recientemente y se están separando del paraguas general de las bases de datos NoSQL, formando una categoría propia. Se han vuelto populares porque miles de profesionales del software y profesionales de los datos han visto que los grafos son la mejor forma de guardar y consultar estructuras de datos

crecientemente complejas e interconectadas. El Dr. Roy Marsten escribió en marzo 2021 que la Teoría de Grafos es un enfoque clave para entender y apalancar el *Big Data* [5].

Los grafos están proporcionando ventajas competitivas también en plataformas como Facebook y Twitter, quienes han usado los grafos sociales para dominar sus mercados, o el mismo Facebook y Google que usan el conocimiento en grafos para impulsar recomendaciones hiperprecisas e hiperpersonalizadas [6].

Otro ejemplo del uso de grafos es eBay, que los usa para computar la entrega rápida de bienes, puerta a puerta, entre compradores y vendedores, escalando su modelo de negocios para incluir la cadena de suministros. eBay observó que antes de utilizar grafos, una consulta demoraba lo mismo, o más, que el tiempo de entrega en los tramos más cortos (15 minutos), mientras que actualmente estas consultas le toman 1/50 fracción de segundo [5].

Es así como hoy en día consultas como "¿dónde están mis llaves?", sin el tipo de semántica que proveen los gráficos, son difíciles de responder. Sin embargo, usando grafos y un poco de procesamiento de lenguaje natural, lleva horas de trabajo (no semanas o meses) implementar un sistema que pueda responder esas consultas.

El poder de una base de datos de grafos es exactamente como tener una mini-web dentro de una aplicación. Se navega la web de nodos a través de relaciones dirigidas, etiquetadas, hasta que se encuentra lo que se necesita, ya sea la ubicación de unas llaves, un viejo amigo del colegio, evidencia a cerca de la eficacia de pruebas clínicas o permisos de acceso a determinados sistemas. El rol de la base de datos de grafos es guardar estos datos de forma segura y hacer consultas rápidas y fácilmente.

3. Bases de datos orientadas a grafos

Desde principios de siglo comenzaron a aparecer y a tomar notoriedad las bases de datos no relacionales (*NoSQL*), dejando de almacenar los datos en filas en tablas, y pasando a almacenarlos en formatos como el *Columnar*, Clave-valor, Documental, o en Grafos.

Cada una de estas bases de datos (en adelante BBDD) están optimizadas para determinados objetivos; en el caso de grafos son usadas para crear modelos sofisticados de datos, de alta fidelidad.

Las BBDD orientadas a grafos permiten almacenar grafos de forma natural y navegar de igual manera. Este tipo de BBDD, a diferencia de las restantes, no necesitan el uso de estructuras intermedias para poder representar los grafos [7].

4. Modelado de grafos

Los grafos son fácilmente entendibles para los humanos, es precisamente eso lo que los hacen poderosos. El cerebro está programado para pensar asociativamente y entender una red de conceptos conectados con otros conceptos.

El modelado más utilizado es el de grafos etiquetado, que se componen de nodos y las relaciones con sus restricciones. Se utilizan convenciones en los nombres para distinguir sus elementos [8].

- **Los Nodos**

  Un nodo típicamente representa una entidad, como una persona, producto, clic, o diagnóstico de un paciente.

  Opcionalmente se le pueden agregar etiquetas a un nodo, indicando el rol del nodo dentro del grafo.

- **Las relaciones**

  Para unir nodos se utilizan relaciones. Las relaciones son dirigidas, y pueden eventualmente tener propiedades asociadas.

  El tipo de relación provee de un predicado, mientras que la dirección de una relación muestra el sujeto y objeto.

- **Restricciones**

  Luego de tener las estructuras básicas, se puede definir cómo evoluciona el grafo.

  Declarando restricciones, se puede pedir a las BBDD orientadas a grafos que se aseguren que ciertas propiedades deben estar presentes en determinados tipos de nodos o etiquetas.

A diferencia de las BBDD tradicionales, donde se requiere un esquema de antemano, en el caso de grafos se espera que los datos crezcan de forma orgánica cuando es posible, y ser restringidos cuando corresponda.

La restricción actúa como un esquema para partes del grafo que requieren un gobierno más fuerte, mientras que otras partes del grafo pueden cambiar de una forma menos restrictiva.

A través de un ejemplo particular [8], en la siguiente figura se modela una familia en grafos. Se muestran 3 nodos etiquetados como "*Person*". Dentro de estos nodos vemos las propiedades *first_name* y *last_name* para Alice, Bob y Charlotte. Se pueden inferir algunas relaciones entre estas personas por sus apellidos en común, pero las relaciones entre ellos lo hacen explícito. Charlotte tiene dos relaciones salientes: MADRE la une con Alice, y PADRE la une con Bob. Estas relaciones son leídas como "la MADRE de Charlotte es Alice", y "El PADRE de Charlotte es Bob".

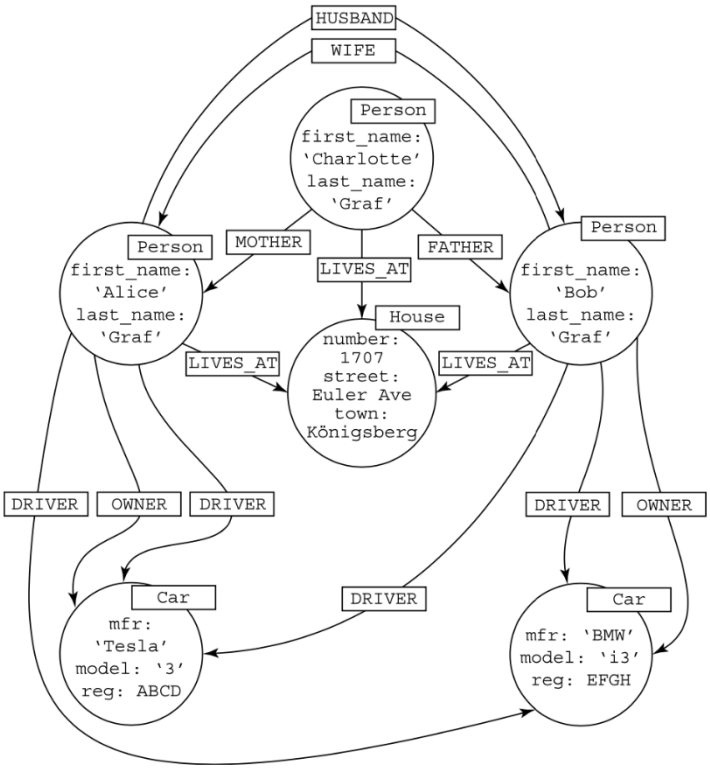


Ilustración 2 Representación de una Familia en grafos

Así mismo se puede ver como cada familia vive de acuerdo con la relación saliente LIVES_AT de cada uno. Siguiendo estas relaciones se puede ver que todos viven en el mismo lugar.

Por otra parte, se observa que Alice y Bob poseen (OWN) un auto, y cada uno es el conductor (DRIVER) de sus propios vehículos. Así que, si Charlotte necesita manejar, puede solicitar tanto a su madre o su padre y puede tomar cualquier auto.

Seguir las líneas entre círculos no parece sofisticado a primera vista, pero es un ejemplo de cómo trabaja Neo4j. Dado un punto de partida, el motor de base de datos persigue punteros dentro del grafo hasta que encuentra la respuesta a las consultas. Por su parte, perseguir punteros es una forma barata y rápida de navegar los datos porque evita hacer combinaciones (*joins*) pesadas y búsquedas lentas y complejas de índices que son comunes en los modelos relacionales.

Este tipo de recorrido tiene su propia denominación: adyacencia libre de índice (*index-free adjacency*), lo cual significa que es posible atravesar desde un nodo a cualquiera de sus vecinos a un costo bajo y constante, y así sucesivamente con el mismo costo por salto. Esto significa que las consultas (*queries*) son proporcionales en tiempo a la cantidad de grafos que la consulta atraviesa o recorre.

Esto permite desacoplar el tiempo de realización (latencia) de una consulta (*query latency*) del tamaño total de la base de datos.

En el ejemplo anterior, las relaciones directas entre nodos etiquetados se denominan "esquema" dentro de la fase de modelado, pero a nivel de implementación, los grafos están poblados por instancias de datos. Habrá tantos nodos "Persona" y "Auto" como tantas personas y autos haya en el sistema.

Se puede generar mayor valor al profundizar el análisis de los grafos siguiendo conexiones transitivas (caminos/*paths*) dentro del grafo. Por ejemplo, amigos de amigos son nuestras mayores influencias, el viaje más rápido en metro puede involucrar cambio de líneas y atravesar varias estaciones, una cuenta *offshore* con movimientos vinculados a un potencial fraude puede levantar sospechas. Todos estos son ejemplos de conexiones indirectas.

La versión inicial del modelo de datos puede ser refinada mediante la realización de consultas provenientes de los expertos en la materia. Generalmente las preguntas involucran varios nodos y relaciones, de los cuales se pueden buscar patrones o atravesar profundamente el grafo para buscar caminos que conectan nodos interesantes. Pueden existir relaciones intermedias entre nodos principales que no resultan tan obvias en el modelado inicial, estos son los datos que queremos encontrar o descubrir, para hacerlo utilizamos preguntas a efectos de refinar y enriquecer la versión inicial del modelo de datos de grafos.

Para considerar un ejemplo adicional [8], con la siguiente tabla de datos con formato CSV, se genera un modelo para pasar esta tabla a un grafo y poder hacer consultas.

Person	Product	Date
Emil Eifrem	Phone	22/05/2020
Emil Eifrem	Computer	20/05/2020
Jim Webber	Computer	08/05/2020
Jim Webber	Bike	12/05/2020
Lance Walter	Fitness equipment	05/05/2020
Lance Walter	Computer	13/05/2020
Lars Nordwall	Boat	21/05/2020
Lars Nordwall	Fitness equipment	11/05/2020
Lisa Hatheway	Phone	27/05/2020
Lisa Hatheway	Boat	10/05/2020
Rik Van Bruggen	Bike	06/05/2020
Rik Van Bruggen	Tablet	27/05/2020

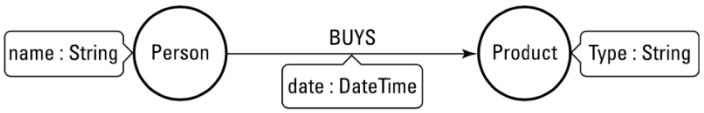


Ilustración 3 Modelación de una tabla en grafo

Para realizar una importación de estos datos a un motor de base de datos orientado a grafos se debe:

- Convertir las entidades *Person* de la primera columna, en nodos *Person.*
- Convertir las entidades *Product* de la segunda columna, en nodos *Product*.
- Crear relaciones BUYS (compra) de *Person* a *Product*, para cada fila en la tabla, y asignar la fecha de la tercera columna a una propiedad de fecha en esa relación.

De esta forma obtenemos el siguiente grafo:

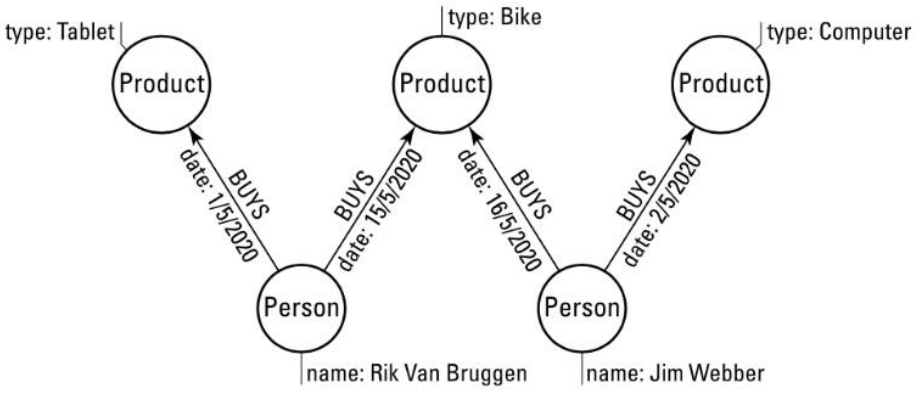


Ilustración 4 Ejemplo de grafo importado

5. Antecedentes de estudios en redes de transporte

En esta sección se repasan algunos de los trabajos relevantes en lo referente a la utilización de bases de datos orientadas a grafos para el análisis y predicción de variables vinculadas a la movilidad y tránsito en una ciudad.

5.1. Modelado de redes de tránsito utilizando *Time Varying Graphs* (TVG)

En los últimos años el estudio de las relaciones existentes entre áreas aparentemente no relacionadas dentro de sistemas dinámicos ha contribuido con la generación de conclusiones y reflexiones más refinadas y cercanas a la realidad [9].

Particularmente evidenciamos esta tendencia en el estudio de las comunicaciones en redes altamente dinámicas, en la explotación de movilidad pasiva, en el uso oportunista o eficiente en redes de transporte y en el análisis de complejas redes dentro del mundo real (neurociencia, biología, transporte, estudios sociales) [9].

Como parte de estos esfuerzos de investigación han surgido una serie de conceptos comunes que terminan estando muy relacionados entre sí. Algunos ejemplos de estos conceptos son: *temporal distance* [9], *reachability time* [10], *information latency* [11], *temporal proximity* [12], *journey* [9], *schedule-conforming path* [13], *time-respecting path* [10], *temporal path* [14].

Una TVG puede notarse como G = (V,E,T), donde V=nodos, E=relaciones y T=tiempo. Asimismo, también definimos como Grafo Subyacente al caso particular de G representado en un momento del tiempo T específico [15].

Existen también lo que se denominan *subgrafos temporales* (G'), los cuales son un subgrupo de grafos de G, restringidos a un marco temporal determinado [15].

- **Distancia**

Existen dos tipos distintos de definición de distancia a considerar en una TVG [15]:

- *La distancia topológica de un nodo u a un nodo v en el tiempo t.*
- *La distancia temporal de un nodo u a un nodo v en el tiempo t.*

Ambos conceptos de distancia son utilizados para una diversa gama de análisis, y particularmente forman parte de la base de distintos algoritmos para computar la distancia más corta o rápida, entre otros.

El uso de los conceptos definidos anteriormente es utilizado para analizar la relación entre la estructura topológica de una red de tránsito y los patrones de tránsito de vehículos en esta red, lo cual resulta crítico para desarrollar una analítica que sea sensible al tiempo y que permita elaborar sistemas de recomendación, tanto para la optimización del tiempo de los usuarios como para la mejora de la infraestructura de la red, la proyección de eventos futuros, o la simulación de escenarios [16].

La estructura topológica de una red de tránsito representa elementos del tránsito tales como calles, caminos, autopistas, estaciones de tren, líneas de metro, vías marítimas, estaciones y paradas. Mientras que las características de movilidad son generadas por los elementos dinámicos dentro de la red tales como buses, autos, trenes, bicicletas, peatones.

La explotación de las relaciones entre estos elementos requiere que los expertos en transporte manejen cantidades masivas de datos que varían a lo largo del tiempo (Lin & Ban, 2013). Asimismo este no es el único desafío sino que también resulta complejo el modelado de estas relaciones a lo largo del tiempo y el espacio [16].

En una red de tránsito la TVG puede contener nodos espaciales, nodos temporales y nodos espacio-temporales. Los nodos espaciales en el grafo cuentan con coordenadas espaciales, de forma que pueden ser ubicados en un mapa (por ejemplo, una parada de ómnibus). Los nodos temporales son nodos con *timestamps*, sin propiedades espaciales (por ejemplo, un viaje). Los nodos espacio-temporales son nodos que tienen propiedades espaciales y temporales en un grafo de tránsito (por ejemplo, subidas o bajadas concretas dentro de un recorrido específico). Generalmente en el tránsito los ejes espacio-temporales son dirigidos entre dos nodos espacio-temporales, y cuentan con atributos temporales así como un peso espacial [16].

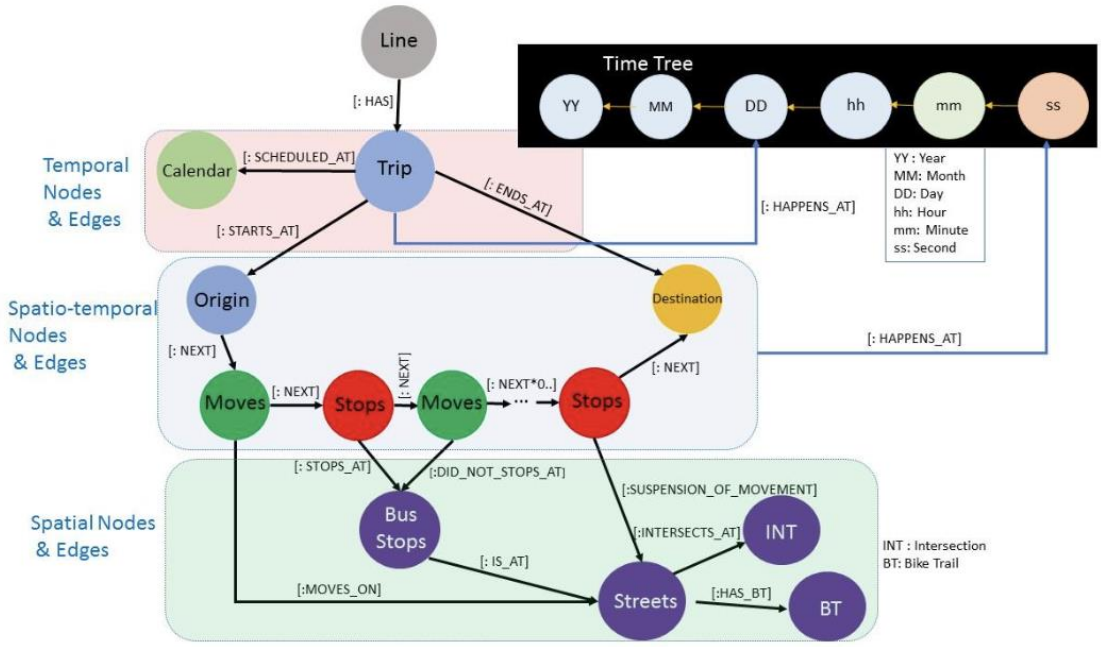


Ilustración 5 Modelado en Grafos de un TVG para una red de transito *[16]*

La Ilustración 5 muestra un modelo lógico de Grafos de una red de tránsito TVG. En este modelo los objetos del grafo están representados por sus respectivas etiquetas y agrupamientos semánticos. Las entidades de la red de tránsito están representadas como nodos con sus relaciones y vínculos. Los nodos están indexados espacial y temporalmente para permitir métricas que ponderen la distancia y la dependencia del tiempo en los análisis que se realicen sobre el grafo [16].

- **Métricas de la red** [16]

- Conectividad

La conectividad es uno de los beneficios clave de una base de datos orientada a grafos por sobre una base de datos relacional. Un grafo se dice que está conectado si existe un camino que atraviesa todos los pares de vértices del grafo. Se dice que un grafo está desconectado cuando existen múltiples vértices y ejes desconectados entre sí. Por ejemplo, en el contexto de las TVG, en una red de tránsito los viajes en ómnibus de origen a destino a lo largo del tiempo son de particular interés.

- Camino (*shortest path* y *longest path*)

Un camino en un grafo es una secuencia de nodos con la propiedad de que cada par de nodos consecutivos está conectado en la secuencia mediante un eje o relación. Se recorre el camino a través de los ejes del grafo desde el nodo origen al nodo destino. Puede ser, por ejemplo, un recorrido de ómnibus concreto, con su origen y destino, atravesando distintas paradas en determinados momentos del tiempo y lugar.

- Centralidad (*degree, betweenness y PageRank*)

La centralidad es una medida relativa de importancia en la red. Existen distintas medidas de importancia asociadas a la centralidad:

Degree centrality: El grado (degree) de un nodo es el número de conexiones que el nodo tiene en la red. Cuanto mayor es el número de conexiones, más influyente resulta dicho nodo en la red. Existe un algoritmo de *Degree Centrality* para medir esta característica de los nodos.

Betweenness centrality: esta métrica mide el número de caminos más cortos de un nodo entre otros dos nodos de la red. Es una medida importante porque puede ser utilizada para servir como un *broker* de servicios en una red.

PageRank centrality: fue utilizado inicialmente por Google [4] para rankear la importancia de los resultados de búsqueda en internet. Esta medida de centralidad asigna un puntaje a los nodos de una red en función de la cantidad de conexiones con nodos de alto grado. Su resultado mide el grado de importancia de un nodo en relación con sus conexiones.

- *Network Diameter* y *Network Density*

El diámetro de una red es el camino más corto más largo (*longest shortest path*). La densidad de la red es una medida de cuán cerca está el grafo de ser un grafo completo (aquel grafo donde cada uno de los pares de nodos existentes en el grafo están conectados entre sí por una única arista). La densidad puede ser utilizada en distintos momentos del tiempo para medir momentos pico dentro de la red.

5.2. Graph Neural Networks (GNN) para predicción de demanda de transporte público

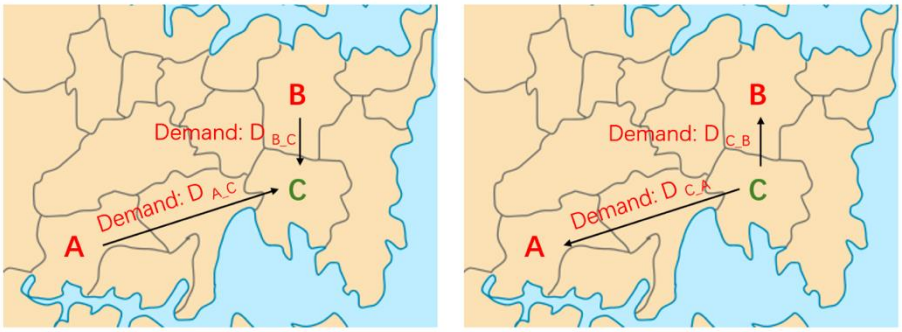
Mientras que los trabajos realizados sobre TVG nos aportan una gran analítica descriptiva y la posibilidad de utilización de algoritmos para detectar patrones de comportamiento y elementos destacados en la red a lo largo del tiempo, la utilización de redes neuronales aplicadas a grafos nos permite proyectar de forma robusta la demanda futura, basada en análisis de series temporales con datos de tiempo y lugar [17].

En particular analizamos un trabajo realizado sobre el uso de tarjetas de tránsito inteligente (Opal Card) de Greater Sydney [17], lo cual guarda un paralelismo muy interesante con nuestro caso de estudio sobre STM en Montevideo, basado en los datos capturados sobre el uso de tarjetas STM.

Los patrones de demanda y movilidad del tránsito varían a lo largo del tiempo y presentan grandes niveles de incertidumbre. Entenderlos, proyectarlos e incorporar la incertidumbre en la demanda resulta muy importante para determinar de forma óptima por ejemplo el tamaño de una flota de buses, tamaño de los vehículos, líneas de transporte público, cronograma y horarios de funcionamiento de las redes.

Por otra parte, esta incertidumbre en el tránsito y la demanda puede generar congestionamiento en determinados rangos horarios, incrementando mayor consumo de combustible y otros efectos secundarios sobre el funcionamiento de la ciudad.

Este estudio [17] propone un modelo probabilístico de redes convulcionales de grafos para proyectar la demanda origen-destino en el sistema de transporte público, dentro de un intervalo de confianza probabilístico. Aplica un método bayesiano para capturar la incertidumbre del modelo y producir un intervalo de demanda predecida. Esto resulta una herramienta poderosa para incorporar la variabilidad de la demanda y ayudar con los problemas operacionales para la toma de decisiones. Utiliza datos reales a gran escala y demuestra la efectividad del método.



(a) Morning Demand (b) Evening Demand

Ilustración 6 Regiones y demanda OD

En el artículo [17] se realiza una partición de la ciudad por códigos postales. Esto permite sacar conclusiones para que las autoridades puedan tomar determinaciones sobre diferentes áreas, y realizar un análisis más racional en función de la actividad desarrollada en esa zona (por ejemplo, en zonas industriales o rurales).

La Ilustración 6 muestra las regiones A y B que son residenciales y la región C que es industrial. Esto significa que tanto A como B tienen atributos similares entre sí, pero diferentes a C. Por la mañana, en las horas pico, la demanda por transporte de las zonas A a C y B a C es elevada. Mientras que por la tarde pasa lo contrario. Esta demanda puede cuantificarse como la cantidad de viajes realizados de un origen a un destino.

Observamos que los resultados de la predicción de la demanda a los que llega el trabajo referenciado guardan una estrecha similitud con los resultados obtenidos en nuestro trabajo sobre la prueba de concepto realizada con los datos de STM, presentados más adelante.

Utilizando una tecnología y modelo de datos con un menor grado de sofisticación o mayor simpleza (algoritmos de *machine learning* como K-Means) sin trabajar sobre la base de una GNN, arribamos a resultados y conclusiones similares (a muy alto nivel) para lo que refiere al comportamiento del tránsito en una ciudad en determinados momentos del tiempo.

Sin embargo, debemos considerar que esta comparación no es estrictamente justa ya que en nuestro trabajo estamos realizando un análisis que podría considerarse más descriptivo sobre los datos observados, mientras que el caso de [17] el trabajo se enfoca en un mecanismo de predicción, con un mayor grado de rigurosidad estadística.

6. Introducción al caso de estudio

La finalidad principal de este trabajo es estudiar los conceptos y desarrollos tecnológicos en torno a las BBDD orientadas a grafos y aplicar algunos de ellos para entender, comparar y validar su nivel de importancia y usabilidad. Al momento de elegir el caso de estudio que se desarrollará en este trabajo, se prioriza el hecho de trabajar sobre datos reales vinculados a la actividad en Uruguay, buscando trabajar en algo que genere mayor impacto o aplicabilidad en el medio local.

6.1. Sistema de Transporte Metropolitano (STM)

Considerando que el modelado de tránsito y transporte ha demostrado ser de utilidad en la aplicación de tecnologías de grafos, se logró identificar que los *datasets* publicados por la Intendencia de Montevideo resultaban por demás atractivos para nuestro trabajo.

Luego de analizar en detalle los datos de STM publicados en la Agencia de Gobierno Electrónico, Sociedad, Información y Conocimiento (de ahora en más, AGESIC) se procedió a establecer contacto con personal de la Intendencia de Montevideo (de ahora en más, IM) para preguntar sobre necesidades puntuales que posiblemente hayan tenido y que puedan ser resueltas mediante analítica en grafos, para entender los datos disponibles públicamente y para saber cómo solicitar información adicional para el desarrollo de este trabajo final.

Se solicitó a la IM información más concreta como lugares de destino (o bajadas) y tiempos e información de rutas para complementar la información previamente analizada.

6.2. Análisis de información STM

Según la información obtenida de STM, se considera una ***parada*** a la localización geográfica y física donde los ómnibus pueden parar para el ascenso y descenso de pasajeros. Dichas paradas están ubicadas en calles particulares de la ciudad y al agruparse conforman una ***línea***, que no es más que una ruta particular, con un determinado origen y destino, por la que transitan los ómnibus. Cada línea puede variar ya sea en sus orígenes o destinos o en las paradas en las que el ómnibus se detiene, dando lugar a un ***código de variante***.

Se llama ***recorrido*** al desplazamiento particular de un ómnibus a través de una línea o variante. Es decir, un ómnibus realiza un recorrido cuando se desplaza por una línea o variante determinada a una hora específica, transportando sus respectivos pasajeros. Es así como un ***pasajero*** simboliza a cada individuo que asciende o desciende de un ómnibus para ser transportado realizando un viaje en particular. El ***viaje*** consiste entonces en el movimiento individual de un pasajero, a una hora específica, desde una parada de origen hasta una parada destino.

La información inicial obtenida de la IM consiste en dos archivos principales:

6.2.1. Información de viajes

Archivo csv que contiene la información de todos los viajes registrados en el STM[1].

id_viaje	identifica de forma única a un viaje dentro del mes. Definiendo por viaje, todos los tramos realizados por el/los pasajeros con un único pago
numero_evento_recorrido	al desplazamiento particular de un ómnibus a través de una línea o variante
con_tarjeta	0 = viajes sin tarjeta, 1= viajes con tarjeta
fecha_evento	fecha hora en la cual se registra el tramo del viaje
tipo_viaje	código del tipo de viaje considerando el tipo de viaje y el grupo de usuario (ejemplo, un estudiante A por defecto realiza un viaje 1 Hora, el cual se considera como un viaje EST. A)
descripcion_tipo_viaje	descripción del tipo de viaje
grupo_usuario	código del grupo de usuario

[1] https://catalogodatos.gub.uy/dataset/intendencia-montevideo-viajes-realizados-en-los-omnibus-del-stm

descripcion_grupo_usuario	descripción del grupo de usuario (ej Estudiante, Jubilado, Usuario Corriente)
grupo_usuario_especifico	código del subgrupo dentro del grupo de usuario
descripcion_grupo_usuario_espe	descripción grupo usuario especifico (ej Estudiante A, Jubilado B)
ordinal_de_tramo	para viajes con tarjeta, ordinal del tramo dentro del viaje
cantidad_pasajeros	cantidad de personas que realizan el tramo
codigo_parada_origen	codigo de la parada de ascenso
cod_empresa	código de empresa a la cual pertenece el ómnibus
descrip_empresa	descripción de la empresa transportista
linea_codigo	código de la línea
dsc_linea	nombre público de la línea
sevar_codigo	código de la variante
fecha_evento	fecha hora en la cual se estima finaliza el tramo del viaje
codigo_parada_bajada	código de la parada estimada de descenso

Ilustración 7 Contenido de información de viajes STM

6.2.2. Información de recorridos

Es un csv que contiene las rutas específicas que debe transitar un ómnibus cuando se desplaza por una línea o variante. Es decir, contiene el conjunto de paradas ordenadas (ordinal) que debe recorrer el ómnibus cuando sale. Esta información es importante porque no se puede

realizar un análisis basándose sólo en información de viajes. Los viajes contienen únicamente información de origen y destino de un determinado pasajero, por lo que es necesario identificar las paradas que recorrió dicho pasajero en su viaje a través del recorrido preestablecido para la línea o variante sobre la que el pasajero realizó su viaje. La siguiente ilustración muestra, para la variante de línea (COD_VARIAN) número 8, el orden (ORDINAL) de las paradas que recorre (COD_UBIC_P)

COD_VARIAN	ORDINAL	COD_UBIC_P
8	1	3413
8	2	3009
8	3	3010
8	4	3011
8	5	4839

Ilustración 8 Ejemplo de recorrido

6.2.3. Información con la geolocalización de las paradas en formato Shapefile

Para este trabajo un dato fundamental fue la geolocalización de las paradas las cuales también se encuentran disponibles en el sitio web de AGESIC[2]. En Python se utilizaron las bibliotecas "Shapefile" para leer dicho formato y poder tenerlo disponible en un *dataframe* de Pandas. Y luego se utilizó la biblioteca "pyproj" y la función "Proj" para poder traducir las coordenadas del formato "UTM 21s" a coordenadas de GPS para una más sencilla importación desde Neo4j

Cod_ubic_p	Código de la ubicación de parada
Cod_varian	Código de la variante de línea de ómnibus (refiere v_uptu_lsv)
Ordinal	Número correlativo de la parada en el trayecto de la variante
Calle	Nombre de la calle sobre la que se ubica la parada
Cod_calle1	Código de la Calle según nomenclator oficial de Montevideo
Esquina	Nombre de la esquina más próxima
Cod_calle1	Código de la Esquina según nomenclator oficial de Montevideo
X	Coordenada X de la ubicación (SIRGAS2000 UTM 21s)
Y	Coordenada Y de la ubicación (SIRGAS2000 UTM 21s)

Ilustración 9 Geolocalización de paradas en STM

[2] https://catalogodatos.gub.uy/dataset/intendencia-montevideo-transporte-colectivo-paradas-y-puntos-de-control

7. Implementación de la solución para el caso de estudio STM

7.1. Modelado de datos STM

Relacionando la información obtenida de STM con la teoría de grafos y los conceptos implementados en Neo4j, las siguientes definiciones componen el modelo implementado:

- Nodos: en este caso de estudio se intentó asignar como nodo aquellos elementos del sistema de transporte relacionados con la infraestructura de la red (paradas, recorridos) como a los elementos que transitan esa red (viajes).
    - Viajes: nodos que representan los elementos registrados en el STM, en definitiva, es cada registro del viaje que hace una persona en el STM y que se captura cuando se lee la tarjeta del pasajero en su ascenso.
    - Recorridos: nodos que representan una estructura de recorrido específica. Por ejemplo, puede ser el trayecto que recorre una línea determinada de ómnibus a una hora en particular.
    - Paradas: nodos que representan cada una de las paradas del sistema. Se crean como nodos ya que cada parada en sí sola representa información relevante, tales como los eventos de ascenso y descenso, y adicionalmente al ser parte de un recorrido otorga información relevante de caminos.
- Relaciones: son los elementos que conectan los viajes, recorridos y paradas de cierta manera.
    - Los viajes se hacen “En” determinados recorridos
    - Un recorrido pasa por una parada en un “Horario” determinado
    - Las paradas están multi-conectadas a través de cada “idviaje” que pasa entre cada una de ellas

La ilustración 10 y 11 muestra cómo quedó el modelado en Neo4j

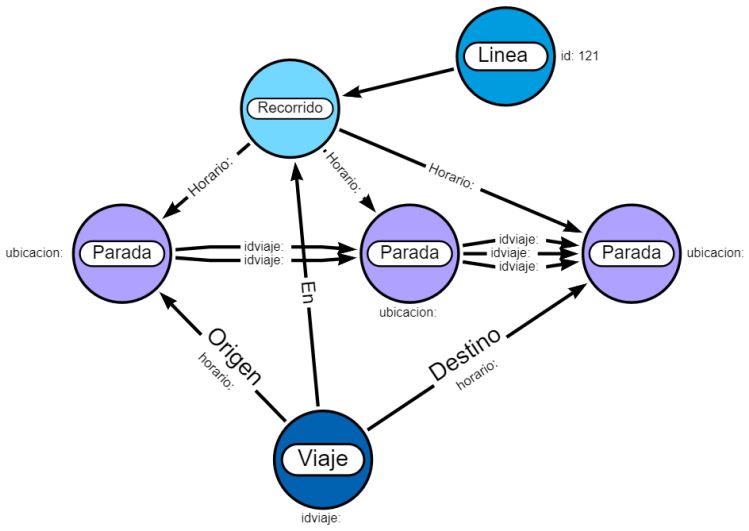


Ilustración 10 Modelado de grafos para STM

Ilustración 11 Relación generada a partir del modelo

7.2. Data wrangling y feature engineering

La información obtenida de la IM se trabajó en una Jupyter Notebook con el fin de crear la información con la estructura necesaria para ser importada en Neo4j (archivos con información de nodos, archivos con información de relaciones) siguiendo las siguientes generalidades:

- Importación de librerías especialmente Spark y Pandas
- Conectar la Jupyter Notebook a los archivos csv enviados por la IM.
- Filtro de información a 1 solo día de registros (el 14 de octubre de 2019 como día base y el 14 de octubre de 2020 para solo efectos de realizar una comparación pre y post pandemia) para procesar los datos sin inconvenientes técnicos, dados los recursos tecnológicos disponibles.
- Modificar los datos de tal forma que se pueda tener un registro de cada parada dentro de ruta que realiza un viaje, inicialmente se tiene solo un registro con la parada de

subida y de bajada. Se realizó una iteración por la cual a cada uno de los viajes se le asignó de manera ordenada todas las paradas que contiene su recorrido.

- Conversión de los datos de geolocalización de las paradas de un formato de Shapefile a coordenadas de GPS, con el fin que la importación desde Noe4j sea más sencilla (sin necesidad de utilizar bibliotecas adicionales dentro de Neo4j)
- Crear las estructuras de datos para ser compatibles con los requisitos para la importación en Neo4j.
- Generar el output de los archivos a importar en Neoj4.

7.3. Importación de datos STM a Neo4j

La importación de los datos generados como outputs en la Jupyter Notebook comienza con la creación de una base de datos nueva dentro de Neoj4. Se aumenta el nivel de uso permitido de memoria que Neoj4 puede utilizar, de tal manera que procese más rápidamente.

Se importaron los nodos y relaciones generadas en la Jupyter Notebook a través de la consola con la herramienta "neo4j-admin import" de Neo4j. Se eligió esta metodología debido a que las recomendaciones sugerían que las cargas iniciales se hicieran mediante un *bulk* en la consola. Y luego algunas cargas accesorias se importaron directo con la función "Load CSV" del lenguaje Cypher[3].

[3] https://neo4j.com/graphacademy/training-importing-data-40/01-import-40-overview-importing/

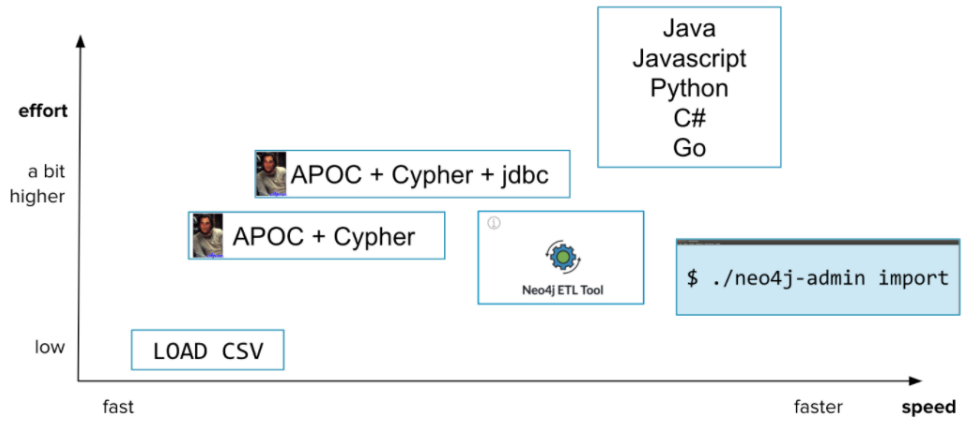


Ilustración 12 Comparativo opciones para importar data en Neo4j[4]

7.4. Configuración inicial de los datos en Neo4j

La siguiente es la consulta utilizada para cargar la ubicación de las paradas en Neoj4 browser:

LOAD CSV WITH HEADERS FROM "file:///ubicacion_paradas.csv" AS row
MERGE (paradas:Paradas {id:row.COD_UBIC_P})
SET paradas.latitude = toFloat(row.lat),
paradas.longitude = toFloat(row.lon)

Resultado: Set 9540 properties, completed after 18527 ms.

Con la siguiente consulta se modificaron las fechas de String a DateTime:

MATCH ()-[r:EN]->() WITH r, apoc.date.parse(r.fecha_evento,'s',"yyyy-MM-dd HH:mm:ss") as t SET r.fecha = datetime({epochSeconds:t})

Resultado: Set 41648 properties, completed after 3261 ms.

Para verificar el cambio en las fechas, se consulta el DataType con la siguiente consulta:

MATCH p=()-[r:EN]->() RETURN apoc.meta.type(r.fecha)
apoc.meta.type(r.fecha)

Resultado: "ZonedDateTime"

Con la siguiente consulta se agregan las fechas a los viajes individuales:

[4] https://neo4j.com/graphacademy/training-importing-data-40/01-import-40-overview-importing/

MATCH (a:Recorrido)-[x:EN]->(b:Paradas)-[r:ViajeIndividual]-
(c:Paradas) WHERE a.numero_evento_recorrido=r.numero_evento_recorrido SET r.fecha_i= x.fecha

Resultado: Set 1370061 properties, completed after 129543 ms.

Se agregan restricciones a la información importada para asegurar su unicidad y generar un indexado que hará más rápidas futuras consultas[5], mediante el siguiente código:

```
CREATE CONSTRAINT constraint_Parada IF NOT EXISTS
ON (n:Paradas)
ASSERT n.id IS UNIQUE

CREATE CONSTRAINT constraint_Recorrido IF NOT EXISTS
ON (n:Recorrido)
ASSERT n.numero_evento_recorrido IS UNIQUE

CREATE CONSTRAINT constraint_Viajes IF NOT EXISTS
ON (n:Viajes)
ASSERT n.id_viaje IS UNIQUE
```

Se carga información adicional para clasificar las horas en momentos del día, con el fin de, en análisis posteriores, agregar la información a nivel de momentos del día

```
LOAD CSV WITH HEADERS FROM "file:///momentosdia.csv" AS row
CREATE (:Hora{hora_id:row.hora})-[:mapeotiempo]->(:Momento{momento_dia:row.momento})

MATCH (b:Paradas)-[r:ViajeIndividual]-
(c:Paradas) WITH r, toString(r.fecha_i.hour) AS hora MATCH (:Hora{hora_id:hora})-[:mapeotiempo]-
>(j:Momento) SET r.momento= j.momento_dia
```

Resultado: Set 1862548 properties, completed after 82103 ms.

[5] https://neo4j.com/docs/cypher-manual/current/constraints/

8. Explotación de datos STM con Cypher

En esta sección se utilizaron tres herramientas para visualizar el resultado de las consultas que pueden ser fácilmente instaladas desde Neo4j Desktop y que se conectan automáticamente a la base de datos activa al iniciarlas:

- **Browser Neo4j[6]:** es una herramienta enfocada en desarrolladores que permite ejecutar consultas en el lenguaje de *Cypher* y visualizar los resultados ya sea como nodos y relaciones dibujadas, *Json* o tablas. Es la interfaz que por defecto se utiliza en todos los ambientes de Neo4j (Neo4j Server, Desktop)
- **Neo4j Bloom[7]:** es una aplicación de exploración de grafos fácil de utilizar para, de manera interactiva y muy visual, interactuar con los grafos en Neo4j
- **Neomap:** es una aplicación para visualizar nodos con atributos geoespaciales (latitud y longitud) en un mapa

8.1. Consultas de negocio

a) Dadas dos paradas no continuas (1480 y 1482) en la calle 4461 (Av. Millán) contar cuántos viajes pasan entre esas paradas por hora y mostrar en una tabla ordenado por hora:

```
neo4j$ MATCH(a:Paradas{id:'1480'})-
       [r:ViajeIndividual*1..2{COD_CALLE:'4461'}]-
       (:Paradas{id:'1482'}) UNWIND r as ee RETURN count(distinct(ee.id_
       viaje_final)) as Cantidad_Viajes, toInteger(ee.fecha_i.hour) as h
       ora ORDER BY hora
```

Ilustración 13 Consulta de negocio 1

[6] https://neo4j.com/docs/browser-manual/current/

[7] https://neo4j.com/product/bloom/

Cantidad_Viajes	hora
2	0
66	5
66	6
310	7
527	8
348	9
255	10
183	11
262	12
270	13
198	14
201	15
154	16
161	17
139	18
89	19
26	20
15	21
26	22
22	23

Ilustración 14 Resultado consulta de negocio 1

Aquí se ve que el mayor pico se da a las 8 am lo cual hace sentido ya que son paradas que están en un recorrido que va hacia el centro y a esa hora de la mañana las personas se trasladan desde zonas más residenciales a zonas más concentradas de servicios

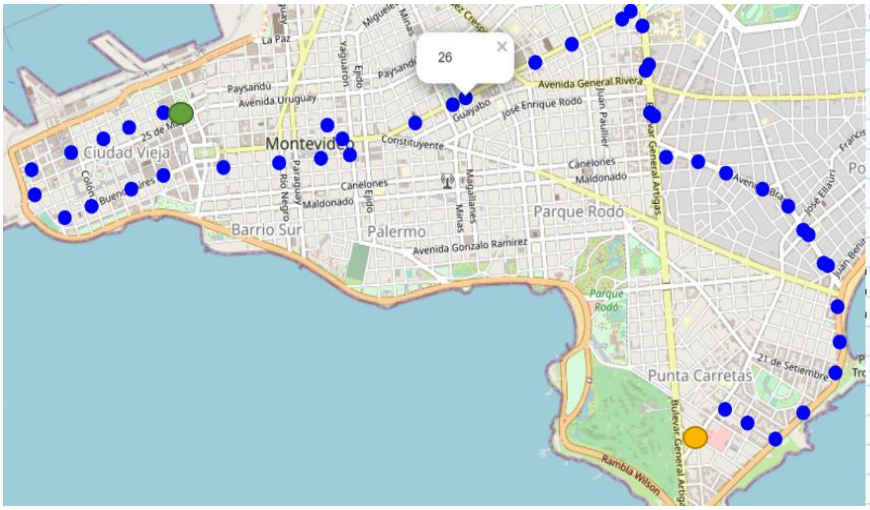
b) Dado un recorrido ('24726368568') particular, mostrar la carga por parada de ese recorrido

```
neo4j$ MATCH (b:Paradas)-
       [r:ViajeIndividual{numero_evento_recorrido:'24726368568'}]-
       (c:Paradas) UNWIND r as ee RETURN
       count(distinct(ee.id_viaje_final)) as Cantidad_Viajes,
       toInteger(b.id) as Parada ORDER BY Parada
```

Ilustración 15 Consulta de negocio 2

La siguiente ilustración muestra en el mapa cada parada (puntos azules) del recorrido #24726368568 de la línea 121, con origen en el punto naranja y destino en el punto verde. El

tooltip muestra la carga de la parada. Adicionalmente, se muestra la misma información en forma de tabla.



Cantidad_Viajes	Parada
48	556
21	574
29	576
31	578
34	579
50	2110
39	2112
47	3180
20	3361
20	3362
20	3363
18	3364
18	3365
16	3366
14	3367
9	3381

Ilustración 16 Resultado consulta de negocio 2

c) Utilizar Bloom para graficar un recorrido y resaltar los tramos con mayor o menor cantidad de viajes

Para hacer esta consulta, primero se guarda en Bloom la consulta que genera el grafo del recorrido

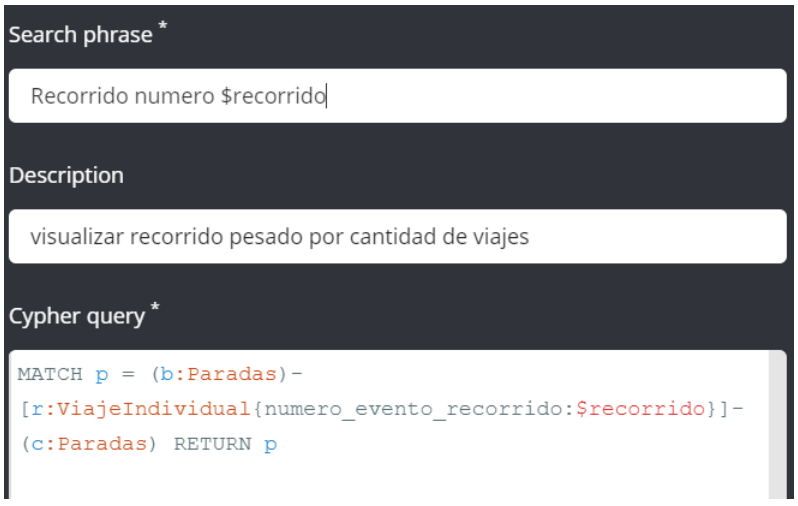


Ilustración 17 Guardar la consulta en Bloom

Luego, se elige la consulta guardada en las opciones de ítems a graficar:

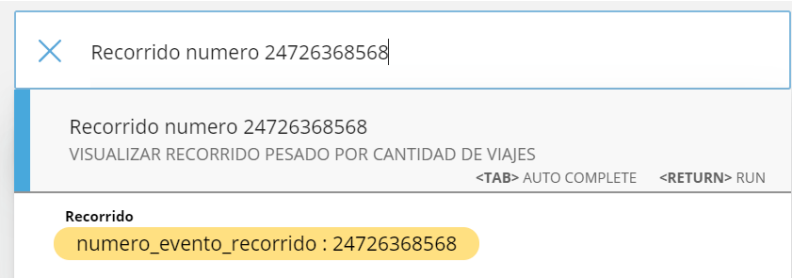


Ilustración 18 Seleccionar la consulta para graficar

Se obtiene el siguiente grafo representativo donde los nodos violetas son las paradas y el ancho de las relaciones representan la carga de viajes dentro del tramo.

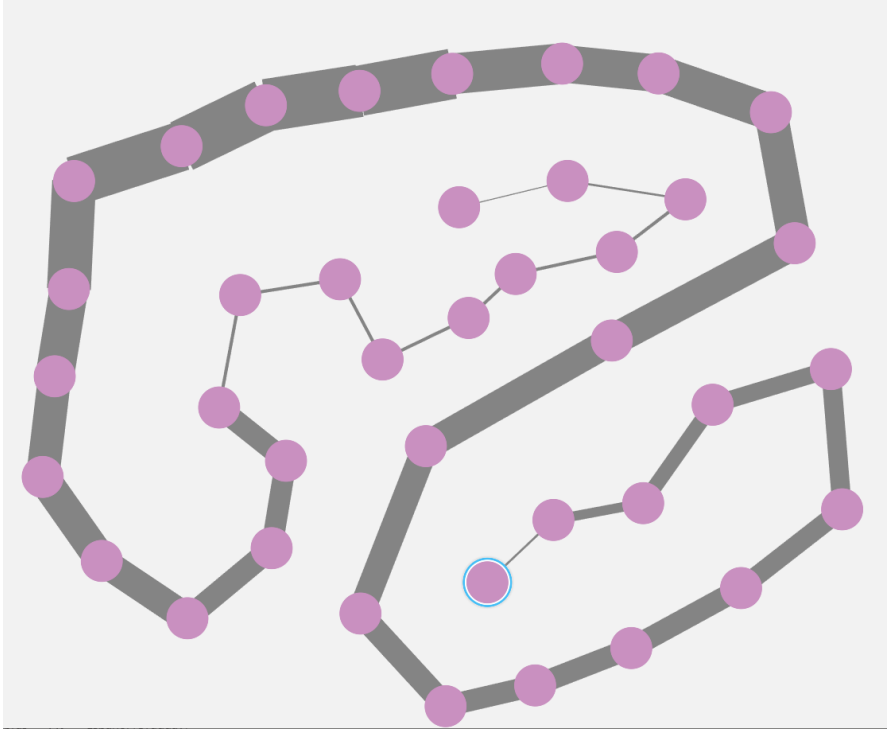


Ilustración 19 Gráfico de consulta de negocio 3

La misma consulta esta vez fue ejecutada desde Bloom, donde la visualización del grafo también es muy gráfica. En este trabajo se están utilizando herramientas estándares (Bloom, NeoMap, Neo4j Browser) que la comunidad e ingenieros de Neo4j pusieron a disposición de los usuarios con una interfaz ya configurada con la base de datos. Con una interfaz gráfica desarrollada a medida, que pudiera juntar las visualizaciones de los puntos 2 y 3 la experiencia del usuario sería aun mejor donde en mismo mapa se podría visualizar las conexiones entre las pardas con un grosor pesado por la cantidad de pasajeros.

d) Para una línea dada (121), mostrar la carga de viajes por hora y por parada. El resultado de la consulta para que sea más descriptivo se grafica en un mapa de calor. De esta manera se puede ver a qué horas y en qué parte del recorrido se genera la mayor concentración de pasajeros.

```
MATCH (a:Recorrido {linea_id:'121'})-[x:EN]→(b:Paradas)-
[r:ViajeIndividual]-(c:Paradas) WHERE
a.numero_evento_recorrido=r.numero_evento_recorrido UNWIND r as
ee RETURN count(distinct(ee.id_viaje_final)) as
Cantidad_Viajes, b.id as Parada, ee.fecha_i.hour
```

Ilustración 20 Consulta de negocio 4

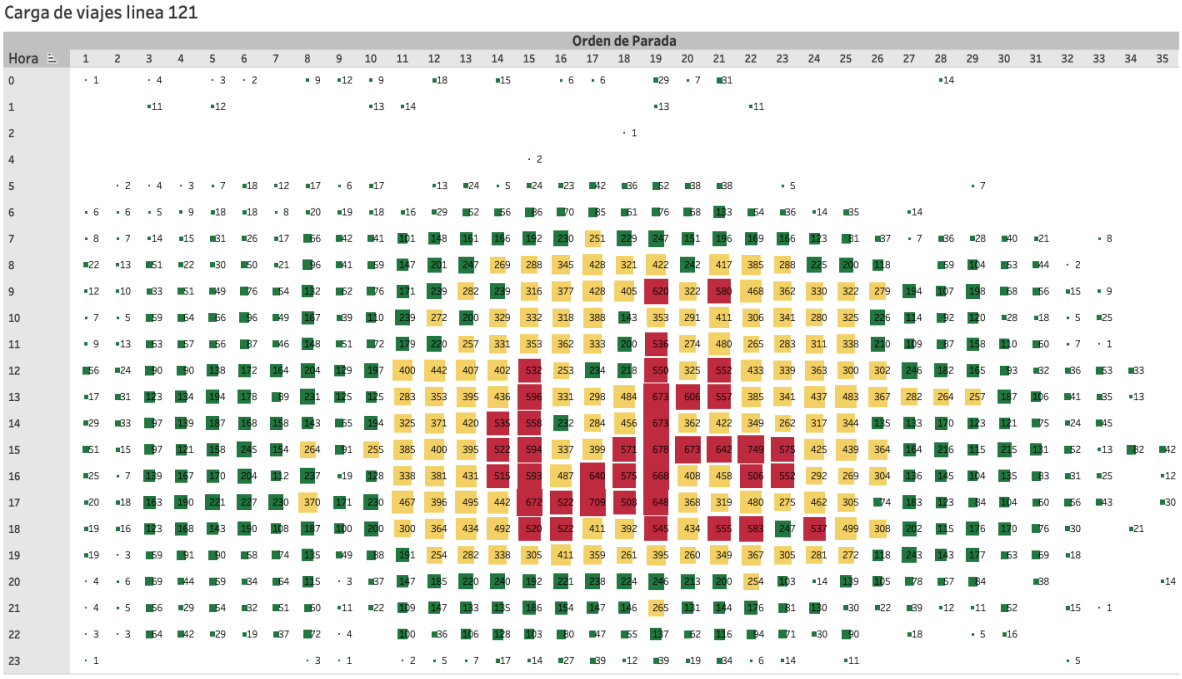


Ilustración 21 Resultado consulta de negocio 4

La ilustración 21 muestra en un mapa de calor (rojo mayor carga de viajes y verde menor carga) para las intersecciones de horarios y tramo (orden de parada) para la línea 121, donde se evidencia que los horarios pico son entre las 12:00 y las 18:00 hs, y los tramos pico se concentran en el medio del recorrido, que coincide con las paradas ubicadas en el centro de Montevideo.

Esta consulta puede ser muy útil en cuanto a las decisiones de por ejemplo en que paradas podrían comenzar y finalizar los recorridos (nuevas variantes de recorrido) dependiendo las horas del día.

e) Dada una calle (código 4461 Av. Millán), mostrar la carga de viajes durante los diferentes momentos del día.

Previamente a ello creamos una nueva relación (arista) entre las paradas llamada “Calle” que dentro de sus propiedades ya agrupa la cantidad de viajes que pasan entre las paradas en cuestión por los diferentes momentos del día.

```
neo4j$ MATCH(a:Paradas)-[r:ViajeIndividual]-(c:Paradas) WHERE r.momento is not null WITH a,c,
       count(distinct(r.id_viaje_final)) as cantidad, r.momento as m, r. COD_CALLE as Calle MERGE
       (a)-[:Calle{Cantidad_Viajes:cantidad,momento:m,calle:Calle}]-(c)
```

```
neo4j$ MATCH (a:Paradas)-[r:Calle{calle:'4461'}]-(b:Paradas) RETURN
       sum(r.Cantidad_Viajes) as Cantidad_Viajes, r.momento as
       momento, a.id as Parada
```

Ilustración 22 Consulta de negocio 5

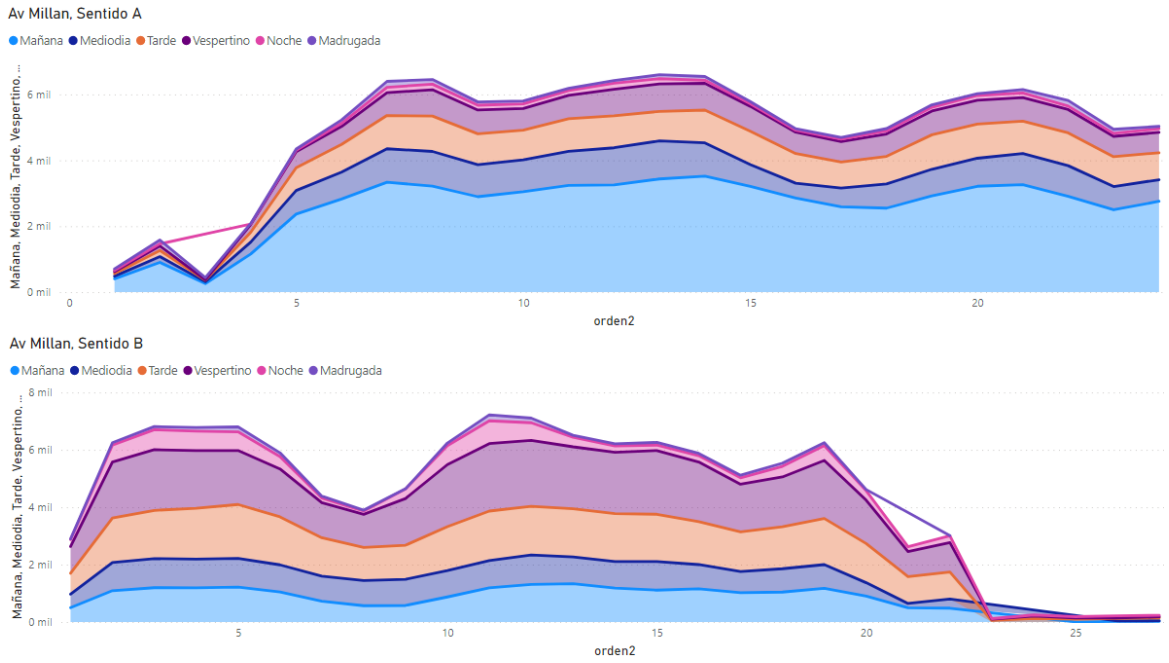


Ilustración 23 Resultado consulta de negocio 5

En la ilustración 23, el eje X corresponde al orden de la parada en el recorrido y el eje Y muestra la cantidad de viajes para cada momento del día. La figura muestra que para el sentido A el

rango horario de mañana concentra la mayor cantidad de viajes, mientras que en el sentido contrario los rangos horarios tienen una carga más pareja, sobresaliendo el horario vespertino. Estas conclusiones hacen sentido con la realidad ya que la calle Millán conecta con el centro de Montevideo, y están alineadas con el hecho que durante la mañana hay mayor cantidad de viajes con dirección al centro que en los otros horarios, mientras que la cantidad de viajes en este momento del día no son tan altos en el sentido contrario y el horario vespertino es donde más cantidad de viajes regresan desde el centro.

Esta consulta puede ser muy útil por ejemplo para decisiones de por ejemplo para calles y horarios en las cuales se observan cargas muy elevadas de viajes, utilizar unidades de ómnibus con mayor capacidad de pasajeros, por ejemplo, ómnibus articulados.

9. Aplicación de algoritmos al caso de estudio

9.1. Algoritmos de centralidad o de importancia de los nodos

Los algoritmos de centralidad son utilizados para entender el rol particular de nodos en un grafo y su importancia en esa red. Son útiles porque identifican los nodos más importantes y no ayudan a entender dinámicas de grupo (Hodler & Needham, 2019).

9.1.1. PageRank

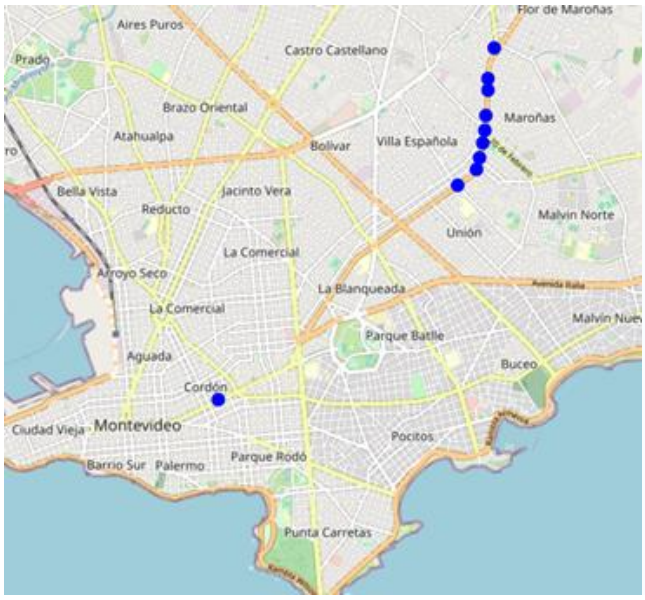
PageRank es el algoritmo más conocido para medir la centralidad. Mide la influencia transitiva de los nodos. Pagerank considera la influencia de los vecinos de un nodo y de sus vecinos. Por ejemplo, tener unos pocos amigos muy influyentes, te hace más poderoso que tener muchos amigos poco influyentes. El cómputo del algoritmo se genera aleatoriamente atravesando el grafo y contando la cantidad de veces por la cual un nodo es atravesado en esos paseos.

En el caso de estudio se corrió el algoritmo sobre el total de las paradas como nodos y como relaciones tomamos los viajes individuales que atraviesan las sucesivas paradas (relaciones). El resultado se puede interpretar como una buena medida de importancia de las paradas en cuanto a su utilización y posición estratégica dentro de la red del STM.

A continuación, el resultado del Top 10 en cuanto al score otorgado por el algoritmo. Donde la mayoría de las paradas pertenecen a la zona de la Av. 8 de Octubre antes de llegar al Intercambiador Belloni. Esa zona puede ser interpretada como un Hub dentro de la red, conectando distintas zonas del área metropolitana.

```
CALL gds.pageRank.stream({
  nodeProjection: 'Paradas',
  relationshipProjection: {
    relType: {type: 'ViajeIndividual', orientation: 'NATURAL', properties: {}}},
  relationshipWeightProperty: null, dampingFactor: 0.85, maxIterations: 20})
  YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
RETURN node.id, round(score,3) as PageRank ORDER BY PageRank DESC
LIMIT 10
```

Ilustración 24 Código Page Rank con STM



	node.id	PageRank
1	"2546"	4.785
2	"2547"	4.172
3	"2548"	3.881
4	"3216"	3.799
5	"3217"	3.581
6	"3215"	3.403
7	"2549"	3.323
8	"4013"	3.287
9	"6194"	3.178
10	"3219"	3.075

Ilustración 25 Resultado Pagerank para STM

9.2. Algoritmos de detección de comunidades

La formación de comunidades es común en todo tipo de redes e identificarlas es esencial para evaluar comportamientos de grupo y fenómenos emergentes.

En el caso de estudio, las comunidades más naturales a estudiar surgen a partir de la cercanía entre las paradas donde los viajes tienen su origen y su destino.

El objetivo de esta sección es generar visualizaciones que muestren de una manera gráfica entre qué zonas de Montevideo se mueven los mayores flujos de pasajeros del STM durante los distintos momentos del día (mañana, tarde, noche, etc). Con dicho fin, el primer paso fue agrupar las paradas por su cercanía. Se estudiaron dos aproximaciones que se presentan en las secciones siguientes.

9.2.1. Clustering con el algoritmo Louvain

El algoritmo de Louvain encuentra Clusters comparando la densidad de cada comunidad y le asigna nodos a los diferentes grupos. Louvain cuantifica cuán bien un nodo fue asignado a un grupo mirando la densidad de conexiones dentro de un Cluster en comparación a un promedio

o a una muestra aleatoria. Computacionalmente es muy rápido y escala bien (Hodler & Needham, 2019).

Para implementarlo en el caso de estudio se tomó un inverso de la distancia entre los nodos como medida (peso) de cercanía (así lo sugiere el artículo en el que se basa este trabajo).

```
CALL gds.graph.create.cypher( 'clustering',
'MATCH (i:Paradas) RETURN id(i) AS id',
'MATCH (i1:Paradas), (i2:Paradas)
WHERE distance(i1.ubicacion, i2.ubicacion) < 210 AND id(i1) <> id(i2)
RETURN id(i1) AS source, id(i2) AS target,
(distance(i1.ubicacion, i2.ubicacion)) ^(-2) AS similarity')
```

```
CALL gds.louvain.stream('clustering', {relationshipWeightProperty:'similarity'})
YIELD nodeId, communityId
RETURN communityId, count(nodeId) as Cant_Nodos ORDER BY Cant_Nodos DESC
```

Ilustración 26 Código de clusterización con Louvain

El resultado de la agrupación es el siguiente:

	communityId	Cant_Nodos
1	2923	112
2	426	69
3	2768	63
4	2735	49
5	2696	48
6	256	48

Ilustración 27 Resultado de clusterización Louvain

Luego de correr el algoritmo se observa que se generaron comunidades (communityId) con demasiados nodos, lo cual dificulta la separación por zonas geográficas identificables. A continuación, se prueba con K-means ya que permite parametrizar la cantidad de comunidades.

9.2.2. K-means

La implementación de k-means no se encuentra dentro la biblioteca de "Graph Data Science" de algoritmos que ofrece Neo4j, pero se encontró un artículo muy interesante en Medium[8], el cual describe el método para implementar k-means con el lenguaje Cypher. La agrupación se hará en función a una de propiedades de los nodos "Paradas", su ubicación.

Dicha propiedad fue grabada con la función nativa de Cypher "Point()", por lo cual Neo4j lo reconoce como un punto en el mapa y permite calcular distancias (función "Distance") entre dichos puntos (expresada en metros).

```
neo4j$ MATCH (p:Paradas) SET p.ubicacion=point({longitude: p.longitude,latitude: p.latitude})
```

Ilustración 28 Asignación de propiedad de geo-ubicación a las paradas

Para implementar este algoritmo de Machine Learning no supervisado se realizaron los siguientes pasos:

- Crear los centroides tomando paradas aleatorias

Se pensó que *clusters* con un promedio de 15 paradas pueden ser representativos de una zona, por lo cual, partiendo de un total de 4798 paradas, se parametrizó el algoritmo para que genere 320 *clusters* (4798/320 = 15 paradas promedio por *cluster)*. Luego, dependiendo de las distancias y la concentración de cada zona, van a existir *clusters* con más o menos paradas.

```
1 MATCH (i:Paradas) WITH i, rand() AS sortOrder ORDER BY sortOrder
2 LIMIT 320 CREATE (c:Centroid) SET c.ubicacion = i.ubicacion,
  c.iterations = 0 RETURN *
```

Ilustración 29 Paso 1 de K-means

- Asignarle a cada centroide un número de Cluster

```
1 MATCH (c:Centroid) WITH collect(c) AS centroids
2 UNWIND range(0, 320) AS num
3 SET (centroids[num]).clusterNumber = num + 1RETURN centroids[num]
```

Ilustración 30 Paso 2 de K-means

[8] https://medium.com/neo4j/k-means-clustering-with-neo4j-b0ec54bf0103

- Asignar cada nodo al centroide mas cercano de manera iterativa

```
call apoc.periodic.commit(
"MATCH (i:Paradas), (c:Centroid)
  WITH i, c ORDER BY distance(i.ubicacion, c.ubicacion)
  WITH i, i.clusterNumber AS oldClusterNumber, collect(c) AS
    centroids
  SET i.clusterNumber = centroids[0].clusterNumber
  WITH i, oldClusterNumber
  WHERE i.clusterNumber <> oldClusterNumber
  WITH count(*) AS changedCount
  MATCH (i:Item), (c:Centroid)
  WHERE i.clusterNumber = c.clusterNumber
  WITH changedCount, c, avg(i.ubicacion.x) AS newX,
    avg(i.ubicacion.y) AS newY
  SET c.ubicacion = point({x:newX, y:newY}),
  c.iterations = c.iterations + 1
  RETURN CASE WHEN c.iterations < 20 THEN changedCount ELSE 0 END
  LIMIT 1",{})
```

Ilustración 31 Paso 3 de K-means

El algoritmo que se eligió para agrupar las paradas por cercanía fue k-means, ya que el mismo nos permite controlar la cantidad de clúster y realizar la agrupación en base a la distancia entre las paradas, independiente que haya aristas entre ellas.

El resultado es el siguiente:

```
neo4j$ MATCH (a:Paradas)<-[:DESTINO]-(b:Viajes)-[r:ORIGEN]->
       (c:Paradas) WHERE r.momento IN ['mañana','mediodia'] WITH
       c.clusterNumber as ClusterOrigen,a.clusterNumber as
       ClusterDestino,count(b) as Cant_Viajes MATCH
       (x:Centroid{clusterNumber:ClusterOrigen})
       MATCH(y:Centroid{clusterNumber:ClusterDestino}) MERGE(x)-
       [:Cantidad_Viajes{ Cant_Viajes: Cant_Viajes,momento:'Dia'}]->
       (y)
```

```
neo4j$ MATCH (a:Paradas)←[:DESTINO]-(b:Viajes)-[r:ORIGEN]→
       (c:Paradas) WHERE r.momento IN ['tarde','vespertino','noche']
       WITH c.clusterNumber as ClusterOrigen,a.clusterNumber as
       ClusterDestino,count(b) as Cant_Viajes MATCH
       (x:Centroid{clusterNumber:ClusterOrigen})
       MATCH(y:Centroid{clusterNumber:ClusterDestino}) MERGE(x)-
       [:Cantidad_Viajes{ Cant_Viajes: Cant_Viajes,momento:'Dia'}]→
       (y)
```

```
neo4j$ MATCH (a:Centroid)-[r:Cantidad_Viajes{momento:"Dia"}]→
       (c:Centroid) WHERE r.Cant_Viajes>250 WITH
       toFloat(sum(r.Cant_Viajes))AS Cant_mayor250 MATCH
       (a:Centroid)-[r:Cantidad_Viajes{momento:"Dia"}]→
       (c:Centroid)  WITH Cant_mayor250, toFloat(sum(r.Cant_Viajes))
       as total RETURN round(Cant_mayor250/total*100) as por_ciento
```

Table / Text

	por_ciento
1	7.0

```
neo4j$ MATCH (a:Centroid)-[r:Cantidad_Viajes{momento:"TardeNoche"}]-
       >(c:Centroid) WHERE r.Cant_Viajes>250 WITH
       toFloat(sum(r.Cant_Viajes))AS Cant_mayor250 MATCH
       (a:Centroid)-[r:Cantidad_Viajes{momento:"Dia"}]→
       (c:Centroid)  WITH Cant_mayor250, toFloat(sum(r.Cant_Viajes))
       as total RETURN round(Cant_mayor250/total*100) as por_ciento
```

Table / Text

	por_ciento
1	9.0

Ilustración 32 Código para generar K-means

Ilustración 33 Correr K-means en Bloom

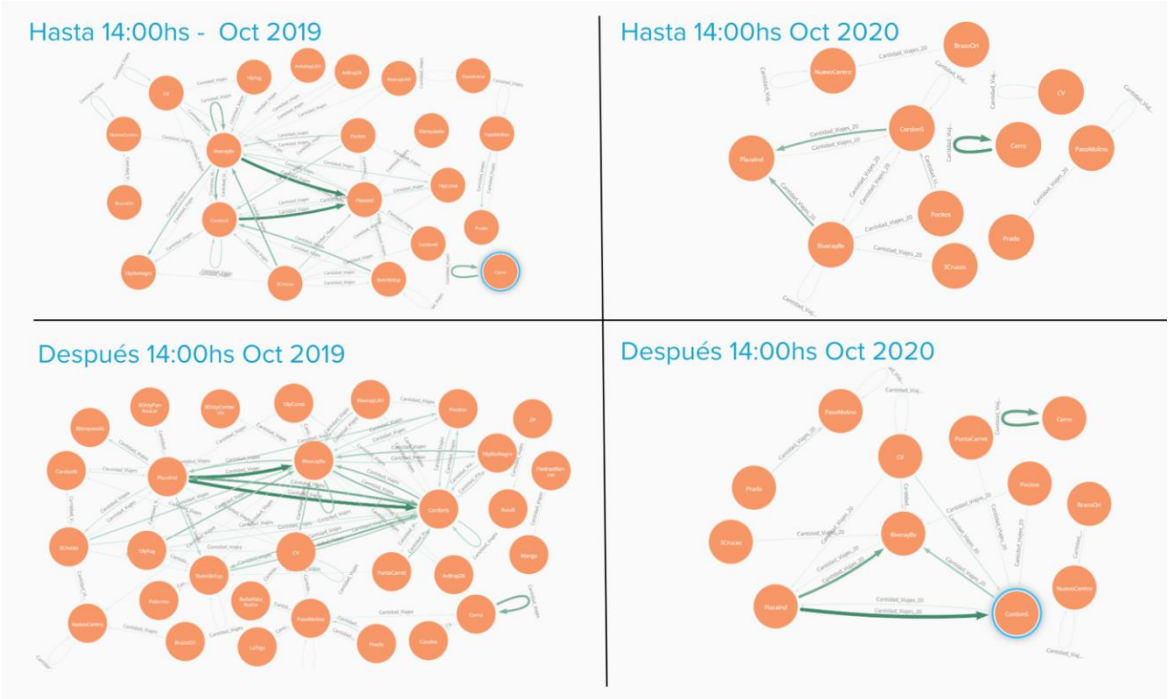


Ilustración 34 Resultado K-means

La ilustración 34 muestra el flujo origen destino de pasajeros (las flechas dirigen el sentido de las relaciones), donde el ancho de las relaciones representa la cantidad de viajes entre cada nodo y los nodos en este caso se convirtieron en *clusters* (zonas de Montevideo) resultantes de correr K-means.

La consulta se ejecutó con un filtro con el fin que solo muestre las relaciones entre nodos que cuentan con más de 250 viajes entre ellos. La ilustración compara dos momentos importantes en el transporte de la ciudad: octubre 2019 como momento previo a la pandemia y octubre 2020 durante la pandemia. Puede verse como la cantidad de *clusters,* entre un momento y otro, baja debido a la reducción de viajes entre nodos, ya que el filtro captura menos nodos. Y también cuáles zonas sufrieron mayor caída de viajes tanto en el origen como destino.

Al margen del claro efecto que visualizamos en la movilidad por causa de la aparición de Covid-19, que implicó una reducción en la movilidad (recordar que el inicio oficial del Covid-19 en Uruguay fue 15 de marzo de 2020 y se extendió al menos durante 18 meses), también se observa un comportamiento distinto según el momento del día (antes o después del mediodía). Como comentamos previamente al analizar el *paper* de [17] en este caso se

alcanza un resultado muy similar de comportamiento, con un flujo en dirección hacia la ciudad en el rango horario previo al mediodía, y un flujo de salida desde la ciudad hacia zonas residenciales por la tarde/noche.

Esta consulta que agrupa y ordena las relaciones de origen y destinos por zonas, puede ser útil en cuanto a decisiones de generar recorridos semidirectos entre las zonas con mayor intercambio.

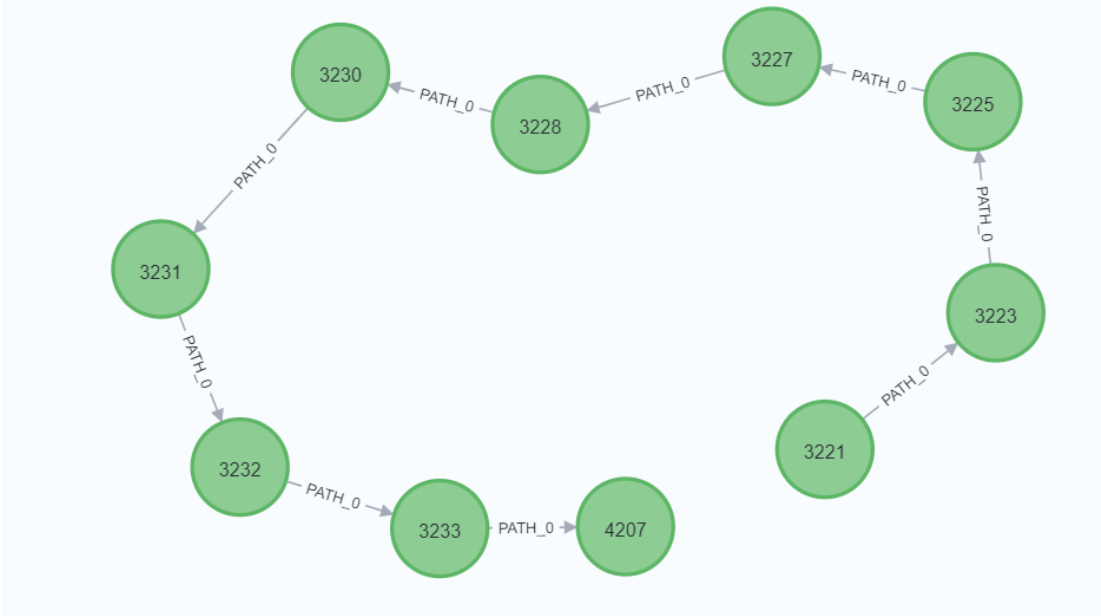
9.3. El camino más corto

Antes de correr el algoritmo es necesario agregar una propiedad a la relación “Calle”, con la distancia entre las paradas continuas, la cual servirá como peso (*relationshipWeigthProperty*) en los algoritmos del camino más corto.

```
1 MATCH (an:Paradas)-[r:Calle]→(bn:Paradas)
2 SET r.distancia = distance(point({longitude: an.longitude,latitude:
  an.latitude}),point({longitude: bn.longitude,latitude: bn.latitude}))
```

```
neo4j$ MATCH(:Viajes{id_viaje_final:'102744912'})-[:ORIGEN]→
       (a:Paradas) MATCH(:Viajes{id_viaje_final:'102744912'})-
       [:DESTINO]→(b:Paradas) CALL
       gds.shortestPath.dijkstra.stream('Rutas',
       {sourceNode:a,targetNode:b,relationshipWeightProperty:'distan
       cia'}) YIELD index, sourceNode, targetNode, totalCost,
       nodeIds, costs, path RETURN index, sourceNode, targetNode,
       totalCost, nodeIds, costs, path
```

Ilustración 35 Consulta para el camino más corto



index	sourceNode	targetNode	totalCost	nodeIds
0	997	6747	2422.375009018736	[997, 999, 1001, 1003, 1004, 1006, 1007, 1008, 1009,

costs
[0.0, 247.60535271100383, 481.09384077700133, 789.1449754085841, 1022.8601303121947, 1437.9290179335871,

Ilustración 36 Resultado del camino más corto

El camino mas corto puede conseguirse a través de varios algoritmos disponibles dentro de Neo4j, de los cuales se destacan Dijsktra y A*, donde Dijsktra puede ser más exacto y A* más rápido. En cuanto al caso de estudio, en principio no le vemos utilidad en cuanto a generar mayor conocimiento para la toma de decisiones, pero si para integrar con otras soluciones a nivel de usuario final de recomendación de rutas.

9.4. Información Geoespacial [18]

A lo largo de este trabajo, hemos utilizado algunas funciones de Cypher que permiten guardar ubicaciones y generar consultas basadas en distancias.

Noe4j ya viene con el tipo de dato para representar un punto en el espacio (*built-in types*) y además tiene un *plugin* para administrar formas más complejas como líneas y polígonos.

Si solo estamos trabajando con puntos con coordenadas, entonces el *build-in* es perfecto ya que permite guardar ubicaciones y generar consultas basadas en distancias.

Sin embargo, para representar puntos espaciales más complejos, como líneas (calles) o polígonos (áreas con límites, zonas, barrios), es necesario utilizar el *plugin* **neo4j-spartial** el cual puede ser bajado de https://github.com/neo4j-contrib/spatial/releases

Neo4j-spartial es una extensión que contiene herramientas para representar y manipular tipos de datos geoespaciales complejos. Dicha extensión permite realizar lo siguiente:

- Importar formatos muy populares como *Shapefile*. Lo cual en el caso de estudio pudiera ser particularmente interesante ya que la IM guarda la información geoespacial en ese formato.
- Aprovechar los índices espaciales para consultas más rápidas.

Los índices espaciales como cualquier otro índice permiten realizar consultas más rápidas evitando tener que realizar operaciones complejas sobre todas las entidades.

Por ejemplo, en una consulta de "intersección", donde estamos buscando los puntos interceptando un polígono complejo, como el que se ve en la figura más abajo, las reglas para evaluar si un punto pertenece a esa forma son bastante complejas. Sin embargo, es mas directo decidir si esos puntos pertenecen al rectángulo pintado de amarillo alrededor de esa figura. Solo los puntos dentro del rectángulo serán evaluados, lo cual reduce la cantidad de operaciones a ser realizadas significativamente. Ese rectángulo es llamado *bounding box* de la figura real.

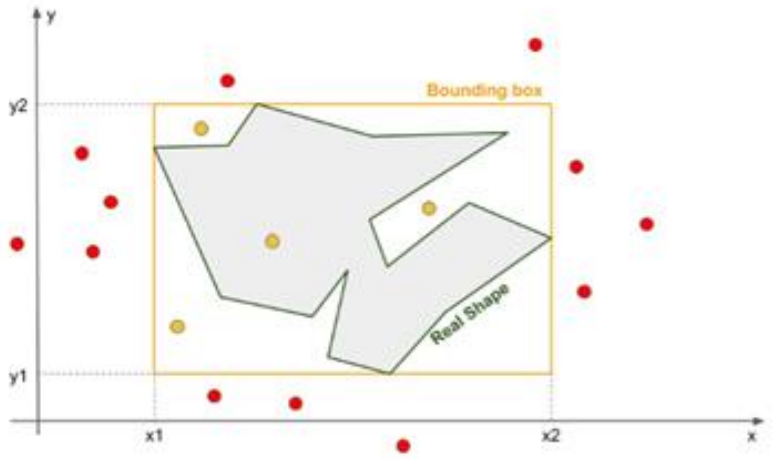


Ilustración 37 Bounding Box y Tamaño Real *[18]*

Capas: una capa (layer) contiene información acerca de un tipo de datos que queremos almacenar. Para crear:

Call spatial.addPointLayer("Capa_Paradas")

Para agregar puntos a la capa:

Match (n:Paradas) Call spatial.addNode("Capa_Paradas") Yield node Return count(node)

Esta operación agrega dos propiedades adicionales al nodo:

- Un atributo *point* como ya hemos utilizado
- Un atributo *bbox* para poder consultar la data de manera indexada como fue explicado anteriormente en esta sección.

Agrega los puntos y también un atributo *bbox* con la finalidad de poder generar consultas indexadas.

Tipos de data espacial:

- *Point* es una simple ubicación

- *Linestring* representa una línea, una colección ordenada de puntos, puede ser usada para representar la forma de una calle o un río por ejemplo.
- *Polygon*, representa un área cerrada, como un edificio, barrio, ciudad o país

Existen estándares para representar estos tipos de geometrías por ejemplo *Well-known Text (WKT)*

Type	**WKT representation**
POINT	POINT (x y)
LINESTRING	LINESTRING (x1 y1, x2 y2, x3 y3, ..., xn yn)
POLYGON	POLYGON (x1 y1, x2 y2, x3 y3, ..., xn yn, x1 y1)

Ilustración 38 Well-known Text (WKT)

Para el caso de estudio, la utilización de polígonos puede ser particularmente útil para delimitar zonas de la ciudad, como lo vemos realizado en este ejemplo de Mahattan en New York

Ilustración 39 Ejemplo Mahattan *[18]*

- Realizar operaciones de topología como “Contiene” o “Intersecciones”

Encontrar nodos a una cierta distancia de uno en particular, por ejemplo, dado una parada, encontrar todas las demás paradas que están a 500 metros o menos de distancia.

Match(p: Paradas {id: '5724'}) Call Spatial.withinDistance(Capa_Paradas), p.point, 0.5) Yield node Return node.ubicacion

Ilustración 40 Nodo Ciudad Vieja

Encontrar nodos que están dentro de una cierta área, por ejemplo, luego de que definamos un área como "Ciudad Vieja" delimitada como un polígono, podríamos realizar una operación para interceptar todos los nodos de etiqueta "Paradas" comprendidos dentro de esa área. La función se invoca: *Call spatial.intersects*

Una propuesta de modelado del caso de estudio teniendo en cuenta los beneficios de esta extensión *spatial* se ilustra a continuación. En la misma se debería agregar una capa geoespacial que contenga el mapa de Montevideo con las esquinas como nodos y las calles como aristas. La misma interactúa con las paradas con operaciones de "contiene" o "intersección". Los beneficios que agrega esta capa son: 1). se pueden crear polígonos para delimitar por zonas o barrios, 2) hace más exacto los cálculos de distancia para algoritmos como el camino más corto y 3) genera indexados por ubicación (bounding box)

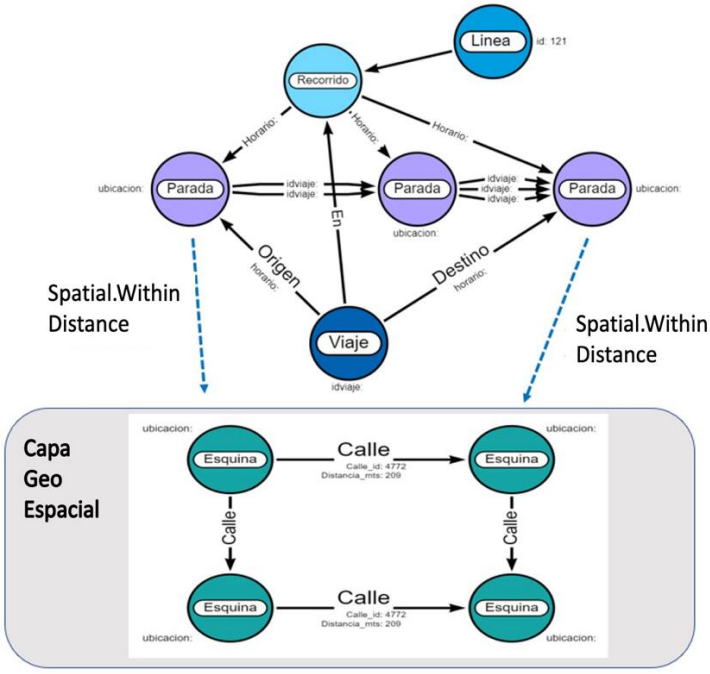


Ilustración 41 Modelado STM con geoespacial

10. Oportunidades de explotación de datos

A continuación, se describen a alto nivel algunos conceptos orientados a integrar y escalar Neo4J en ambientes big data como Apache Spark (que coincide con el *framework* que actualmente utiliza la IM para gestionar sus datos) y en aplicaciones avanzadas de visualización.

10.1. Neo4j y Apache Spark [19]

Apache Spark es actualmente un popular orquestador de datos con una poderosa herramienta de ETL a nivel Big Data. En noviembre de 2020, Neo4j anunció el lanzamiento del conector para Apache Spark que permite mover y modificar datos de forma bidireccional entre estos, introduciendo el ecosistema de Apache Spark a Neoj4 y permitiendo que Neoj4 tenga acceso a las fuentes de datos que ya están conectadas a Spark.

Con esta integración ahora es posible manipular, integrar, extraer, consolidar y/o agregar datos desde Apache Spark u otras fuentes de datos y luego moverlos a Neo4J para hacer procesamiento de grafos.

Usando el *Datasource API* de Apache Spark se puede implementar Neo4j dentro de Spark como si fuese una fuente de datos adicional para obtener un *DataFrame* y editarlo.

```
spark.read.format("org.neo4j.spark.DataSource")
  .option("url", "neo4j://my-cluster:7687")
  .option("labels", "Person")
  .load()
  .show()
```

Ilustración 42 Código de ejemplo para importar nodos de Neoj4 en Spark

La Ilustración 10 muestra cómo importar nodos (“Person”) de Neo4j en Spark, pero más aún, el conector permite trabajar con 3 tipos de objetos en grafos, tanto para lectura como para escritura: nodos, relaciones y consultas. Las consultas de Cypher pueden usarse para expresar algún patrón de grafos dentro de Spark, permitiendo llamar algoritmos especiales de grafos y leer los resultados directamente.

El conector de Spark-Neoj4 es ideal para realizar trabajos de extracción, transformación y carga de datos de tal manera que se puede conectar la información almacenada en distintas

fuentes de datos a pipelines de ingeniería de datos o aprendizaje automático siguiendo este flujo de trabajo:

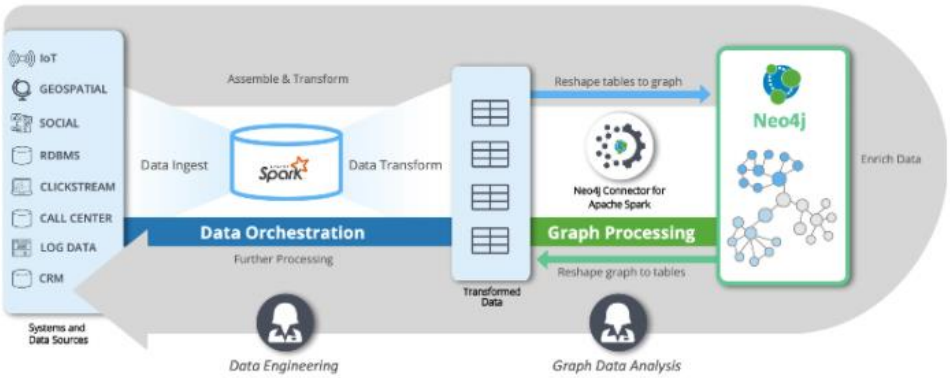


Ilustración 43 Flujo de trabajo de integración entre Neoj4 y Apache Spark

- Ensamble y transformación de datos: conectarse a la fuente de datos y transformarlos con Spark.
- Moldear tablas a grafos: se puede escribir cualquier *DataFrame* en Neoj4 convirtiendo las tablas a grafos. Esto es llamado "*Tables for Labels*"
- Enriquecer los datos: con la información convertida en grafos dentro de Neoj4 pueden generarse nuevas conexiones entre los nodos que le den más valor a los grafos.
- Moldear grafos a tablas: pueden leerse los grafos previamente enriquecidos para usarlos en Spark. Esto es llamado "Labels for Tables"
- Procesamiento adicional: con la información proveniente de Neo4j pueden generarse nuevas perspectivas de información que pueden ser almacenadas en la fuente de datos original.

10.2. Herramientas de visualización de grafos[9]

Sin lugar a duda la visualización de datos permite añadir valor a la analítica de datos de una forma muy rápida y directa, más aun hablando de grafos, donde una vista particular de un grafo puede otorgar conclusiones explícitas, ya sea de nodos de influencias o de detección de comunidades.

[9] https://neo4j.com/developer/tools-graph-visualization/

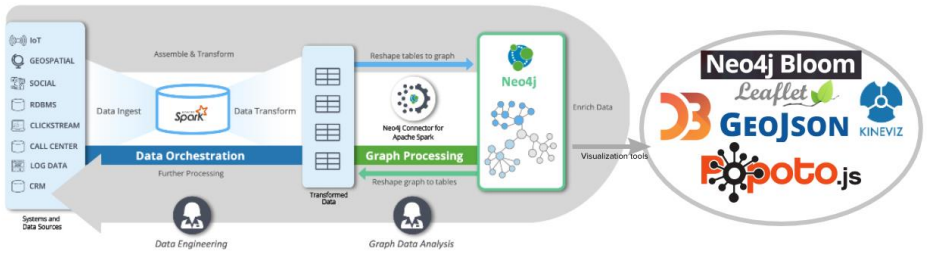


Ilustración 44 Integración Neo4j, Spark y Visualización

En este trabajo práctico se han desarrollado algunas visualizaciones a los grafos de STM usando Bloom, que es una herramienta desarrollada por Neoj4 para la exploración de datos que permite navegar y consultar la información sin tener que utilizar un lenguaje de consulta o de programación específico. Adicional a esta herramienta, Neo4j puede usar 3 tipos de arquitecturas para la visualización de datos:

- Herramientas embebidas con conectores de Neo4j:

  Son herramientas incluidas dentro de Neo4j que pueden ser fácilmente configuradas para ser usadas. Se conectan a alguna instancia de la base de datos de grafos y permiten crear visualizaciones basadas en nodos, relaciones o propiedades, como parte de la interfaz de usuario. Suelen ser librerías gestionadas por la comunidad para soporte y mejoramiento. Ejemplos:

  - Neovis.js
  - Popoto.js

- Herramientas embebidas sin conectores de Neo4j:

  Sirven para agregar visualización de grafos dentro de una aplicación, pero sin conectarse directamente a Neo4j, lo que permite agregar a la visualización datos obtenidos mediante una API conectada a la base de datos, pero con el inconveniente que la información tiene que ser extraída de Neoj4 en el formato que requiera la librería. Ejemplos:

  - D3.js
  - Vis.js
  - Sigma.js

- Herramientas independientes:

  Están diseñadas como aplicaciones independientes que pueden conectarse a Neo4j e interactuar con la información almacenada sin usar códigos, especialmente pensadas para analistas de negocios, científicos de datos o cargos gerenciales. Algunas herramientas requieren licencias comerciales, pero pueden personalizarse para casos de uso particulares. Ejemplos:

  - Bloom by Neo4j
  - GraphXR by Kineviz
  - yFiles by yWorks
  - Linkurious
  - Graphistry
  - Keylines by Cambridge Analytics

En cuanto a problemas de sistemas de transporte como el abordado en este caso de estudio, las herramientas de visualización pueden ser aún mas importantes si incluyen características de geolocalización. Según esto, existen en el mercado herramientas de visualización de grafos con mapas, tales como:

- Leaflet: Es una librería de Javascript para mapas interactivos. Permite crear múltiples capas para ocultar o mostrar y permite, además, crear una capa de mapa base sobre la cual se pueden desarrollar visualizaciones, como relaciones entre distintos puntos de localización, rutas, recorridos, etc.
- GeoJSON: Es un formato estándar abierto para representar información geográfica basada en formato JSON. Incluye la representación de puntos, líneas o polígonos. No solo permite representar entidades del mundo físico, sino que también permite la aplicación de rutas móviles o navegación.

11. Conclusiones y Futuros trabajos

- Se profundizó en los principales conceptos vinculados a bases de datos orientadas a grafos y particularmente en herramientas que resultan de utilidad para gestionar información relacionada al transporte público de la ciudad.
- Se logró aprender la herramienta Neo4j, sus funcionalidades, almacenamiento, consulta, algoritmos y visualización. Además de algunas aplicaciones desarrolladas en la comunidad de Neo4j.
- Se encontró en Neo4j una herramienta que permite procesar un gran volumen de datos, comprobando que es una herramienta eficiente para manejar casos de uso como el transporte, combinado con atributos temporales y geoespaciales, permitiendo hacer consultas que pueden generar valor al usuario de la información, visualmente amigables.
- Se concluyó que el modelado del grafo resulta ser un factor importante a la hora del desarrollo de este caso, ya que define desde el inicio muchos de los pasos desarrollados: el proceso de *data engineering*, la forma de importación, las estructuras de las consultas.
- Se lograron implementar consultas interesantes de forma sencilla usando cypher (carga o densidad de pasajeros por recorrido, por momentos del dia, por tramos) y algoritmos (page rank, camino mas corto, comunidades) que respondieron varias consultas de negocio.
- Se identificaron algunos *next steps* que la IM puede usar para poner en producción los análisis de este trabajo:
    - Conexión con *Spark* y ambiente Hadoop para la ingesta de datos en Neo4j
    - Incorporación de información geográfica de calles y esquinas para enriquecer y hacer más precisas las consultas.
    - Utilización de *Spark* para obtener la información procesada de Neo4j y disponibilizarla en un *pipeline* de aprendizaje automático para hacer análisis predictivos sobre la demanda de transporte.

- Indexar los datos geográficos en capas, por ejemplo (biblioteca *Spatial*), crear polígonos para limitar áreas, lo cual puede agregar valor al análisis además de hacer más rápidas las consultas.
- Conectar la base de Neo4j a una capa de visualización (a través de drivers con *javascript* como d3.js, popoto.js, *Leaflet*) en la cual pueda verse simultáneamente información en el mapa con grafos que muestren la densidad o carga de pasajeros.

12. Referencias bibliográficas

[1] C. Godsil y G. Royle, «Wikipedia,» 2011. [En línea]. Available: https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos.

[2] Redes1, «Wikipedia,» 2013. [En línea]. Available: https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos.

[3] M. Needham y A. Hodler, «Graph Algorithms: Practical Examples in Apache Spark and Neo4j,» Sebastopol, CA, O'Reilly Media, Inc., 2019.

[4] S. Brin y L. Page, «The Anatomy of a Large-Scale Hypertextual Web Search Engine,» *Standford: Computer Science Department.*

[5] E. Eifrem, «Graph Theory: key to understanding big data,» *wired.com,* 2014.

[6] J. Ugander, B. Karrer, L. Backstrom y C. Marlow, «The Anatomy of the Facebook Social Graph,» Palo Alto, CA, 2011.

[7] D. Packer, «A Brief Introduction to Graph Data Platforms,» San Mateo, CA, 2020.

[8] J. Webber y R. Bruggen, «Graph Databases,» Neo4j, San Mateo, CA, 2020.

[9] B. Bui-Xuan, A. Ferreira y A. Jarry, «Computing shortest, fastest, and foremost journeys in dynamic networks,» International Journal of Foundations of Comp, 2003, p. 14(2):267–285.

[10] P. Holme, «Network reachability of real-world contact sequences,» *Physical Review E,* 2005.

[11] G. Kossinets, J. Kleinberg y D. Watts, «The structure of information pathways in a social communication network,» *In Proceedings of 14th ACM International Conference on Knowledge Discovery and Data Mining (KDD),* p. 435–443, 2008.

[12] V. Kostakos, «Temporal graphs,» *Physica A,* p. 388(6):1007–1023, 2009.

[13] K. Berman, «Vulnerability of scheduled networks and a generalization of Menger's Theorem,» *Networks,* p. 28(3):125–134, 1996.

[14] A. Chaintreau, A. Mtibaa, L. Massoulie y C. Diot, «The diameter of opportunistic mobile networks,» *Communications Surveys & Tutorials,* p. 10(3):74–88, 2008.

[15] A. Casteigts, P. Flocchini, W. Quattrociocchi y N. Santoro, «Time- Varying Graphs and Dynamic Networks,» *University of Ottawa,* 2012.

[16] I. Maduako , E. Cavalheri y M. Wachowicz, Exploring the use of time-varying graphs for modelling transit networks, Fredericton, Canadá: University of New Brunswick., 2018.

[17] W. Liu, L. Bai y L. Yao, Graph Neural Network for Robust Public Transit Demand Prediction, IEEE Transactions on Intelligent Transportation Systems, 2020.

[18] E. Scifo , Hands-On. Graphic Analytics with Neo4j, Packt Publishing, 2020.

[19] M. Natarajan y D. Allen, The Great Hookup: Announcing Neo4j Connector for Apache Spar, San Mateo, CA: Neo4j Blog, 2020.

13. Anexos

La Jupyter Notebook con el procedimiento de *data engineering* y archivos complementarios a este documento se pueden encontrar en:

https://github.com/EnriquePeirano/Tesis-Big-Data-Grafos