Modelado de datos Temporales y Espaciales de Meteorología en Bases de Grafos

Timothy López *Facultad de Ingeniería, Universidad de la República*
Montevideo, Uruguay
timothy.lopez@fing.edu.uy
Braulio Ríos *Facultad de Ingeniería, Universidad de la República*
Montevideo, Uruguay
braulio.rios@fing.edu.uy

Resumen

En este trabajo se relevan y comparan distintos modelos y métodos para ejecutar consultas filtrando por información temporal (fecha y hora) y de distancia espacial (latitud y longitud) para una base de grafos creada en neo4j. Se utiliza una parte de los datos meteorológicos de Brasil, para obtener un conjunto de datos manejable localmente en cualquier computadora personal estándar, y se comparan los tiempos de ejecución de las distintas soluciones propuestas, así como la sobrecarga que agregan sobre la base de datos en distintos aspectos. Finalmente se muestran algunos casos de uso simples para ejemplificar la utilidad de hacer eficientes las consultas que se discutieron.

I. Introducción

I-A. Objetivos

El objetivo general de este proyecto es evaluar el funcionamiento de una base de datos de grafos, para trabajar con información estructurada a través de atributos temporales (fecha y hora) y espaciales (latitud y longitud). En este caso, se utilizarán datos de meteorología, pero la intención es que las técnicas y herramientas utilizadas sean generalizables a otros dominios, ya que este tipo de atributos se encuentran habitualmente y suele ser importante poder realizar consultas a través de ellos.

Utilizando el motor de base de datos **neo4j**, se buscará comprender las herramientas y opciones disponibles, así como el impacto que tienen algunas decisiones de diseño sobre el desempeño general. Particularmente se evaluarán los compromisos a asumir entre múltiples dimensiones como: velocidad de las consultas, volumen de datos extra necesario (*overhead*), facilidad para asegurar la integridad de los datos, y complejidad del modelo (para facilitar la intuición de uso y mantenibilidad del mismo).

Es importante resaltar que para encontrar el modelo más eficiente para resolver un problema de negocio determinado, es necesario conocer con cierta precisión los casos de uso y definir de antemano las consultas a realizar. En este proyecto sin embargo, se busca un entendimiento generalizable y aplicable más allá del caso particular a resolver, y en realidad se usan estos datos a modo de guía y se muestran algunas posibles consultas representativas, que permiten entender algunos patrones más generales. Es decir, que el foco no está puesto en resolver un problema de negocio bien definido, sino en encontrar algunas claves para entender los compromisos de cada decisión de diseño.

También es un objetivo de este proyecto, comprender las particularidades a considerar a la hora de utilizar un motor de bases de grafos, que son herramientas relativamente nuevas si se comparan con bases relacionales. Éstas promueven además un enfoque muy distinto en muchos aspectos, lo cual requiere generar una nueva intuición que actualmente no es común entre los desarrolladores. Por lo tanto, también se incluirán comentarios más generales sobre cualquier aspecto que pueda parecer importante a tener en cuenta al evaluar este tipo de soluciones para futuros proyectos.

I-B. Trabajos Relacionados

El modelado de datos espaciales y temporales en bases de datos de grafos ha ocupado a profesionales y especialistas en la disciplina en los últimos años, con el objetivo de poder conseguir el modelo más intuitivo y adecuado para el manejo eficiente de los recursos.

Para el presente trabajo se toman como base diferentes trabajos, artículos y presentaciones relacionadas al modelado de datos espaciales y temporales:

- *Graphs in Time and Space: A Visual Example* [9], presentación expuesta en el *GraphConnect Europe* en Mayo del 2017.
- *Graphing Space and Time* [10], presentación expuesta en el *GraphConnect New York* en Setiembre del 2018.
- *Building Spatial Search Algorithms for Neo4j* [8], presentación expuesta en el *Global Nodes meeting 2019*.
- *An Efficient Graph-Based Spatio-Temporal Indexing Method for Task-Oriented Multi-Modal Scene Data Organization* [5], artículo publicado en Setiembre del 2018 en la *ISPRS International Journal of Geo-Information*.

I-C. Datos utilizados

Se utiliza parte del dataset abierto (licencia GPL-2) publicado en Kaggle [7] con los datos meteorológicos de toda la superficie de Brasil.

Se usará todo el rango de fechas disponibles, desde el 7 de Mayo del 2000, al 30 de Abril de 2021.

Este dataset se encuentra particionado por regiones de Brasil, y en este caso se utilizará sólamente la región cubierta en el archivo `central_west.csv`, que corresponde a mediciones tomadas por 116 estaciones meteorológicas. En total, el dataset completo tiene disponibles 622 estaciones, y la información de cada estación (código, nombre, latitud, longitud, altura, etc) se encuentra separada en el archivo `stations.csv`, que también se utilizará.

No se utiliza el dataset completo porque con esta cantidad de datos se puede trabajar localmente sin necesidad excesiva de almacenamiento en disco o memoria RAM, y a su vez se pueden encontrar consultas que demoran en el orden de segundos a minutos antes de ser optimizadas, y así ilustrar bien el cambio introducido con cada potencial mejoría.

Con los datos utilizados en este proyecto se contará entonces con 622 estaciones, pero sólo 116 de ellas tendrán datos meteorológicos disponibles. El resultado es un dataset con 11,427,120 filas, donde cada una tiene múltiples variables meteorológicas, de las cuales se importarán las siguientes: radiación, temperatura, dirección del viento, velocidad del viento, presión, humedad y precipitaciones. La descripción de cada columna se puede ver en el archivo `columns_description.csv` en [7]. En la siguiente sección se brindan más detalles sobre el código para importar los datos a la base de grafos.

II. Desarrollo

II-A. Carga y preprocesado de datos

II-A1. Reproducibilidad de los experimentos: Si bien existen herramientas gráficas como *Data Importer* para facilitar la carga de archivos CSV en neo4j, se encontró mayor flexibilidad y mejor reproducibilidad al crear scripts en lenguaje Cypher basados en la instrucción `LOAD CSV`, como se pueden ver en el código del proyecto [4], en el directorio `import/`. Esto permite que cualquiera pueda reproducir estos resultados sin necesidad de ninguna aplicación, plugin o librería extra, otra que la instalación estándar de *neo4j Desktop*.

La implementación de estos scripts se realizó en principio siguiendo la guía oficial en [1], con la salvedad de que incluso allí se hace referencia a la instrucción obsoleta `USING PERIODIC COMMIT`, necesaria para la carga de archivos con grandes cantidades de filas como las 11 millones de mediciones. Finalmente, se logró implementar esto utilizando la sintaxis `CALL {} IN TRANSACTIONS`, que es la nueva forma en la que se debería cargar un archivo grande en batches de transacciones, según la descripción oficial de esta cláusula [3].

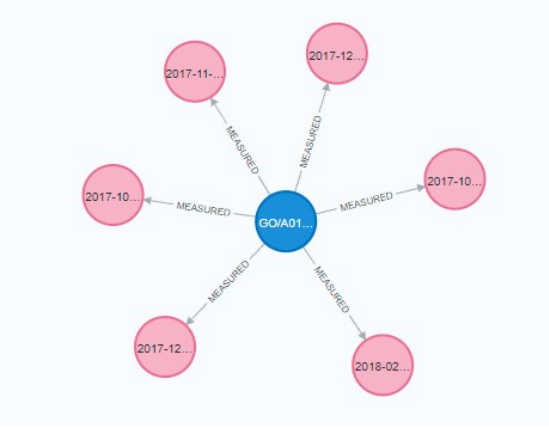

Figura 1: Nodo central `(:Station)` y algunas mediciones vinculadas `(:Measurement)`

II-A2. Consideraciones de calidad de datos: Los scripts implementados para importar los datos en el proyecto [4] transforman cada fila de `stations.csv` en un nodo con etiqueta `(:Station)`, y cada fila de `central_west.csv` en un nodo con etiqueta `(:Measurement)`. Además de los nodos, se crean aristas entre esos nodos, para lo cual se debe conocer a qué estación corresponde cada medida. En un principio, parecía adecuado utilizar la columna `station_code` para hacer el vínculo, sin embargo se encontró que estos códigos no son únicos. Además, la columna `name` de las estaciones tampoco es única, incluso con estaciones en distintos estados que tienen el mismo nombre. Para poder realizar el vínculo entre estaciones y medidas, se necesita una forma de crear un identificador único de cada estación, que también se pueda recrear en las filas de las mediciones. La solución encontrada es concatenar el código del estado federal, con el código de estación y finalmente el nombre de la estación. Por ejemplo, un nombre de estación es `GO/A011/SAO SIMAO`, y así figura en el campo `Station.name` y `Measurement.station_name`, y se construyen aristas de tipo `MEASURED` entre estos nodos como se muestra en la figura 1.

El identificador anterior permite crear un índice único sobre los nodos con etiqueta `(:Station)`. A su vez, para asegurar que cada medición de cada estación es única para un día y hora dados, se creó otro índice *multikey* único sobre los nodos `(:Measurement)`, con las claves conjuntas `(station_name, name)`, donde a su vez `Measurement.name` es un string que contiene fecha y hora (e.j: `2018-08-04T22:00`). No se encontraron problemas de unicidad al crear este índice sobre las mediciones.

II-B. Búsqueda basada en atributos temporales

Una consulta muy habitual en cualquier tipo de base de datos, es la búsqueda de registros para una fecha y hora dados. En meteorología y en otras aplicaciones donde podría ser necesario hacer algún tipo de predicción, y por lo tanto entrenar modelos (estadísticos o de aprendizaje automático), este tipo de consultas suelen ser la base para extraer batches de datos durante el proceso de entrenamiento, y por lo tanto es necesario hacerlas de la forma más eficiente para que sea posible extraer la información necesaria con alta frecuencia, y evitar que la lectura de datos sea el cuello de botella de un sistema que puede ser muy costoso por tiempo de infraestructura.

Por lo tanto, es de gran interés conocer bien los posibles mecanismos y alternativas para realizar consultas de este tipo. Comenzamos por realizar la búsqueda **sin índices** en ningún campo temporal, para poder tener una idea del tiempo que demora el motor en recorrer todos los nodos o aristas de cierto tipo, y ver si algunas decisiones de diseño permiten que este recorrido completo no sea necesario. Además, la búsqueda con índices hace muy difícil la comparativa de tiempos de ejecución ya que para un conjunto de datos de tamaño manejable y estático como en este caso, no tienen contra indicaciones (el espacio en disco y el costo de mantener los índices actualizados no es un problema en este caso).

II-B1. Búsqueda por nodos: La opción más inmediata para realizar la consulta en esta base de grafos es equivalente a lo que haríamos en una base relacional, y consiste en tener campos `(date, time)` (en este caso lo hacemos por separado, aunque podría ser un sólo campo de tipo `datetime`) en los nodos con etiqueta `(:Measurement)`.

Así, la consulta para obtener las mediciones de todas las estaciones en el primer día del año 2019 a las 2:00AM, es simplemente:

```
MATCH (s:Station)-->(m:Measurement)
WHERE m.hour = time("2:00") AND m.date = date("2019-01-01")
RETURN s, m
```

Como no existe un índice para estos campos, esta consulta requiere recorrer los 11 millones de nodos `(:Measurement)` secuencialmente, lo cual resulta en un tiempo de ejecución de casi medio minuto como se puede ver en la tabla I.

II-B2. Búsqueda por aristas: Una variante de la consulta anterior que no debería cambiar demasiado la esencia de la búsqueda, es colocar la información de fecha y hora de las medidas, en las aristas de tipo `[MEASURED]`. Aunque la cantidad de registros a recorrer debería ser la misma, ya que cada medida tiene una arista y por lo tanto existen 11 millones de éstos para comparar, esta opción también fue probada para entender si el motor hace algo distinto al recorrer aristas en vez de nodos:

```
MATCH (s:Station)
      -[:MEASURED{hour: time("2:00"), date:date("2019-01-01")}]->
      (m:Measurement)
RETURN m
```

Por alguna razón, la primera consulta utilizando aristas demora aproximadamente lo mismo que la búsqueda por nodos, pero luego en las siguientes iteraciones el tiempo baja a menos de 20 segundos, como se ve en la figura 2. El tiempo promedio de 5 consultas seguidas se puede ver en el cuadro I.

La razón por la que el tiempo de la segunda consulta en adelante es menor, posiblemente sea que estas aristas se almacenan en la memoria RAM en caché (ver [6]), lo cual hace que la lectura de los registros sea más rápida. Bajo esta hipótesis, la siguiente interrogante es: ¿por qué no sucedía lo mismo en la consulta sobre los nodos?. La respuesta podría ser que, tal como se explica en la referencia citada, los registros se guardan en caché siempre que estos entren en memoria, y aunque el número de nodos y aristas es igual en ambas consultas, los nodos contienen mucho más información en forma de propiedades (todas las mediciones meteorológicas), que tal vez hacen que almacenar todos estos registros en la memoria disponible no sea posible.

Podría explorarse el funcionamiento del caché en mayor profundidad, y cómo permitir que los nodos se guarden en RAM. Sin embargo, lo más sustancial es que aún en este caso sigue siendo necesario el recorrido de todos los registros y por lo tanto ésta no es una consulta eficiente ni siquiera luego de que el motor guarda los registros en caché.

II-B3. Subgrafo temporal: Una variante para mejorar la búsqueda por atributos temporales que es más importante a nivel de modelo, y que además aprovecha mejor el hecho de contar con una base de grafos, es crear nodos específicos para cada posible valor de atributo. Es decir, se crean nodos de tipo `(:Year)`, `(:Month)`, `(:Day)`, `(:Hour)`, donde cada nodo

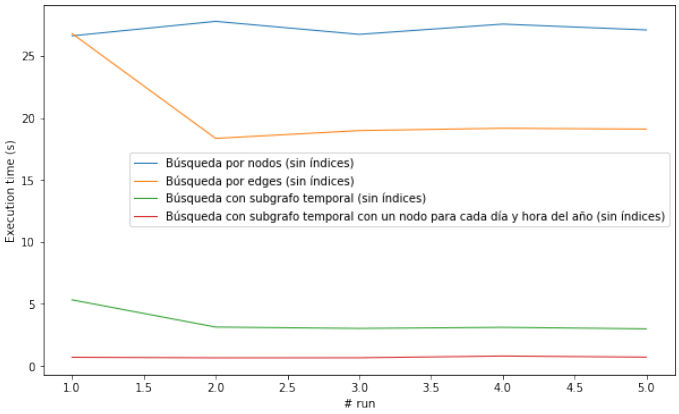

Figura 2: Tiempos de ejecución para n=5 iteraciones seguidas de cada consulta temporal, sin índices

tiene un valor posible (e.j: 12 nodos con valores de meses, 31 nodos de días, etc), y luego se trazan aristas entre éstos y los `(:Measurement)` que tienen cada valor, como se ve en la figura 3.

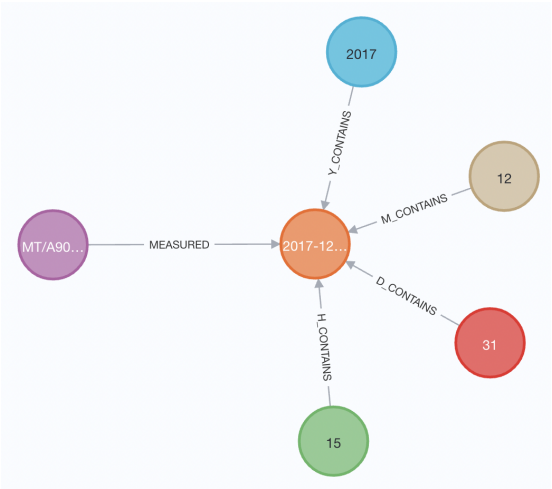

Figura 3: Subgrafo con atributos temporales en forma de nodos (año 2017, mes 12, día 31, hora 15) relacionados a la medida (nodo naranja) para ese instante en una estación dada (nodo lila)

Esto genera un subgrafo por cada valor de atributo que contiene un menor número de medidas, y por lo tanto debería acelerar las consultas al reducir el espacio de búsqueda. Por ejemplo, si buscamos en el subgrafo para el nodo de una cierta hora del día, este tendrá aproximadamente la cantidad total de registros (11 millones) dividido entre 24. Luego, para una fecha y hora dados, el motor tendrá que hacer la intersección entre todos estos subgrafos para dar el resultado, lo cual debería ser eficiente en un motor de bases de grafos. Es decir, la consulta es:

```
MATCH (:Month{name:1})-->(m:Measurement)<--(:Year{name:2019}),
      (m)<--(:Day{name:1}),
      (m)<--(:Hour{name:2})
RETURN m
```

Cuadro I: Tiempos de ejecución promedio de las consultas temporales (fecha y hora) sin índices

Consulta	**Tiempo promedio de ejecución (n=5)**	**Desviación estándar (n=5)**
Búsqueda por nodos	27.1634s	0.4544s (1.67 %)
Búsqueda por aristas	20.4800s	3.1762s (15.51 %)
Búsqueda con subgrafo temporal	3.5167s	0.9064s (25.77 %)
Subgrafo temporal (un nodo por día)	0.6992s	0.0519s (7.43 %)

Y en el cuadro I se observa que el tiempo promedio para ejecutar la consulta baja drásticamente a los 3.5 segundos, además de que parecería hacerse uso del caché para alguno de los subgrafos, porque el tiempo baja luego de la primera consulta (figura 2).

II-B4. Subgrafo temporal con un nodo por día: Finalmente, un siguiente paso a la hora de construír un subgrafo temporal, es generar un nodo para cada posible valor de instante temporal, y relacionarlo a todas las mediciones para ese instante. Es decir, existirá un nodo `(:DateTime)` que represente al día 1/2/2019 a la hora 2am y en este caso tendrá una arista con cada una de las 112 mediciones de las estaciones activas. El espacio de búsqueda una vez identificado el nodo temporal, es de sólo 112 registros. Sin embargo, ahora el subgrafo temporal tiene una cantidad potencialmente muy grande de nodos `(:DateTime)` entre los que buscar, que dependerá del intervalo de tiempo entre los que hay mediciones (en este caso hay 196.416 nodos, correspondientes a todos los días y horas posibles en algo menos de 22 años). La cantidad de aristas sigue siendo de 11 millones, una para cada medición.

Para hacer eficiente la búsqueda entre esta enorme cantidad de nodos temporales, seguimos utilizando los nodos `(:Year)`, `(:Month)`, `(:Day)` y `(:Hour)`, pero esta vez apuntan a los nodos `(:DateTime)` y permiten encontrarlos más rápidamente. Esto agrega un número de aristas no despreciable, de 4X por cada `(:DateTime)` (aunque en realidad podría ser 3X ya que el filtrado por `(:Hour)` es equivalente a buscar entre los 24 nodos que quedan en el subgrafo luego de filtrar por las otras variables).

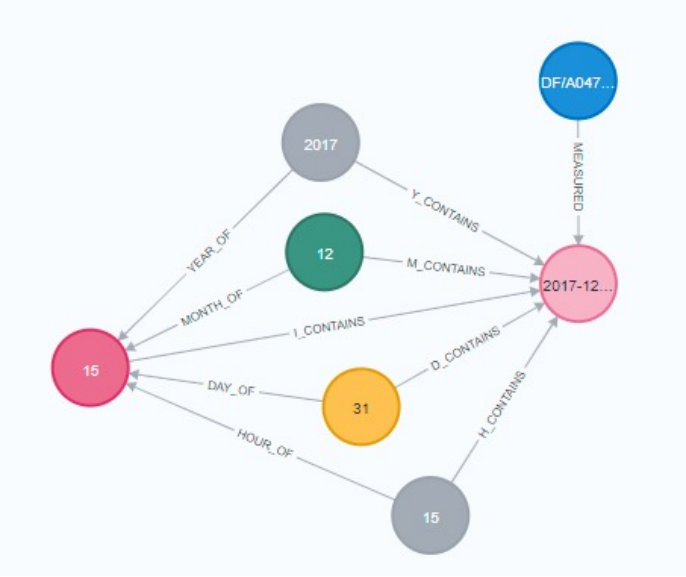

Figura 4: Subgrafo temporal con un nodo para cada posible día y hora de cada posible año.

La figura 4 muestra el modelo resultante, donde las relaciones `Y_CONTAINS`, `M_CONTAINS`, `D_CONTAINS` y `H_CONTAINS`, deberían dejar de usarse, en favor de las `I_CONTAINS` (excepto que se quieran consultar todas las mediciones de cierto atributo, por ejemplo todo el año 2017).

Este tipo de grafo temporal ha sido usado previamente en [5], y de allí surgió también la idea del modelo anterior. A nivel de eficiencia de las consultas, se puede ver en el cuadro I y en la figura 2 que es rápido incluso en los modelos sin índice. Sin embargo, se debe tener en cuenta que genera una cantidad de nodos bastante importante, que según el problema a resolver, se debe evaluar su viabilidad teniendo en cuenta el período de tiempo a cubrir y la resolución temporal que se desee. En este caso, el problema aún es viable a nivel horario, si se deseara una resolución de minutos o segundos, la cantidad de nodos necesaria puede volverse inmanejable con el tiempo.

Finalmente, cabe mencionar que generar las aristas entre las 11 millones de medidas y los casi 200 mil nodos horarios, es una operación muy costosa computacionalmente, que en este caso llevó varias horas y casi vuelve inviable la solución. Esto no sería un problema si el caso de uso fuera agregar datos *en vivo*, secuencialmente con el tiempo (es decir, 112 mediciones una vez por hora). Y a su vez, hay que considerar que pueden existir soluciones intermedias con menor resolución temporal que se adapten mejor según el problema de negocio a resolver, pero parece claro que reducir cada vez más el espacio de búsqueda agregando nodos intermedios que agrupen ciertos registros, mejora la velocidad de las consultas.

II-B5. Búsqueda con índices: Luego de entender como los distintos modelos generan consultas con mayor o menor eficiencia a nivel de espacio de búsqueda, se agregan los índices a los campos temporales y se compara la velocidad de correr exactamente las mismas consultas.

Se crean los siguientes índices que son relevantes para las consultas mostradas anteriormente:

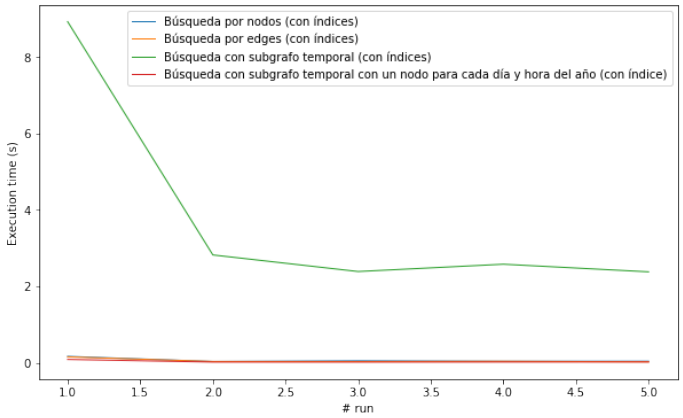

Figura 5: Tiempos de ejecución para n=5 iteraciones seguidas de cada consulta temporal, luego de crear índices sobre los campos de fecha y hora

```
// Índices para búsqueda por nodos
CREATE INDEX meas_hour IF NOT EXISTS FOR (m:Measurement) ON (m.hour)
CREATE INDEX meas_date IF NOT EXISTS FOR (m:Measurement) ON (m.date)

// Búsqueda por edges
CREATE INDEX rel_meas IF NOT EXISTS FOR ()-[r:MEASURED]-() ON (r.date, r.hour)

// Búsqueda de nodos temporales de día y hora DateTime
CREATE INDEX date_time IF NOT EXISTS FOR (d:DateTime) ON (d.hour, d.day, d.month, d.year)
```

Para los nodos de `(:Year)`, `(:Month)`, `(:Day)` y `(:Hour)` no se crean índices, ya que la cantidad de registros con estas etiquetas es muy pequeña y no deberían ser necesarios.

Los resultados de correr nuevamente las consultas luego de la creación de estos índices, se pueden ver en el cuadro II y en la figura 5. Es notorio que los tiempos de todas las consultas bajan, excepto para la búsqueda por subgrafo temporal donde no se usa un nodo por día. En realidad, tiene sentido que esta consulta no cambie demasiado, porque los índices creados no afectan a ninguna de las propiedades involucradas en el filtrado. Por el contrario, la búsqueda por nodos y aristas se reduce en tres órdenes de magnitud, y la búsqueda con el subgrafo con un nodo por día/hora también se reduce (ya que se crea el índice para buscar nodos `(:DateTime)`), y representa la consulta más eficiente en términos de velocidad, llegando a menos de 20ms luego de la primera ejecución.

II-C. Búsqueda por coordenadas espaciales

II-C1. Optimizando la consulta de distancia: La búsqueda por coordenadas en *neo4j* es posible gracias a la incorporación de tipos de datos espaciales a partir de su versión 3.4 [8] y a las funciones espaciales del lenguaje de consultas *Cypher* [2].

En cuanto al tratamiento de los datos espaciales del presente caso, se optó en primera instancia por analizar las distintas consultas sobre los datos de latitud y longitud presentes en las estaciones `(:Station)` utilizando las funciones mencionadas en el párrafo anterior **sin utilizar índices espaciales**.

La función más útil para este caso es la de medición de distancia entre puntos:

```
point.distance(point1, point2)
```

Cuadro II: Tiempos de ejecución promedio de las consultas temporales con índices en los campos de fecha y hora

Consulta	**Tiempo promedio de ejecución (n=5)**	**Desviación estándar (n=5)**
Búsqueda por nodos	0.0650	0.0507s (78.00 %)
Búsqueda por aristas	0.0556	0.0491s (88.33 %)
Búsqueda con subgrafo temporal	3.8157	2.5575s (67.03 %)
Subgrafo temporal (un nodo por día)	0.0311	0.0274s (87.93 %)

Se analiza puntualmente esta función y su complejidad en la ejecución de consultas. A modo de prueba, en primera instancia se consultan las estaciones `(:Station)` a menos de 300 kilómetros de una cierta estación (MS/A756/AGUA) que tengan mediciones `(:Measurement)` para una hora y fecha específica a modo de comprarlas con la medición de la estación de referencia:

```
MATCH (s0:Station{name: "MS/A756/AGUA CLARA"})-->(m0:Measurement)
MATCH (s:Station)-->(m:Measurement{hour: m0.hour, date: m0.date})
WITH s0, s, m0, m, point.distance(point({longitude: s.longitude, latitude: s.latitude}),
point({longitude: s0.longitude, latitude: s0.latitude}))/1000 AS travelDistance
WHERE travelDistance < 300 AND m0.date = date("2019-01-01") AND m0.hour.hour = 2
RETURN s0, s
```

En esta primera aproximación la distancia se calcula entre estaciones y no entre mediciones. Sin embargo, el hecho de que el filtrado por distancia se hace sobre los pares `(:Station) - (:Measurement)` en vez de sólamente entre nodos de `(:Station)`, hace que la distancia se calcule en cada par previo a filtrar por distancia, resultando en una consulta muy lenta, como se detalla en el Cuadro III.

Habiendo notado la baja performance de la consulta anterior se opta por primero filtrar las estaciones por distancia, y luego hacer el matching con con sus mediciones:

```
MATCH (s0:Station{name: "MS/A756/AGUA CLARA"})
MATCH (s:Station)
WITH s0, s, point.distance(point({longitude: s.longitude, latitude: s.latitude}),
point({longitude: s0.longitude, latitude: s0.latitude}))/1000 AS travelDistance
WHERE travelDistance < 300
MATCH (s0)-->(m0:Measurement)
MATCH (s)-->(m:Measurement{date:m0.date, hour: m0.hour})
WHERE m0.date = date("2019-01-01") AND m0.hour.hour = 2
RETURN s0, s
```

En esta segunda aproximación el tiempo de ejecución mejoró considerablemente como lo muestra el Cuadro III. Sin embargo, aún se tiene que ejecutar la distancia entre todas las estaciones (622 nodos), y no sólo entre las que tienen mediciones para ese día y hora.

Como un nuevo intento de mejora, se prueba primero filtrar las mediciones, y luego buscar las estaciones por distancia. Volviendo a una consulta similar a la primera, salvo que se cambia el orden de las sentencias `WITH` / `WHERE`:

```
MATCH (s0:Station{name: "MS/A756/AGUA CLARA"})-->(m0:Measurement)
MATCH (s:Station)-->(m:Measurement{hour: m0.hour, date: m0.date})
WHERE m0.date = date("2019-01-01") AND m0.hour.hour = 2
WITH s0, s, m0, m,
point.distance(point({longitude: s.longitude, latitude: s.latitude}),
point({longitude: s0.longitude, latitude: s0.latitude}))/1000 AS travelDistance
WHERE travelDistance < 300
RETURN s0, s
```

Esta última consulta demora en el orden de los 60ms como se puede apreciar en el Cuadro III. Por lo tanto, aunque a priori no parezca intuitivo aplicar el `WHERE` primero sobre las mediciones (que son muchos más nodos que las estaciones), ejecutar una función de distancia es relativamente costoso, en concordancia a lo expuesto en [8].

II-C2. Modelando distancia entre nodos: En la presentación de Taverner en [8], él propone como método de optimización a nivel de modelo en datos espaciales, crear las aristas entre los nodos con referencia espacial (latitud y longitud) y dentro de estas aristas agregar la propiedad de la distancia calculada previamente. En primera instancia parece una buena solución siempre y cuando el volumen de nodos sea manejable y más aún tomando en cuenta que el dato de distancia entre nodos con referencia espacial no suele ser un valor dinámico.

Habiendo tomado en consideración la propuesta previamente detallada, se procede a crear las aristas entre las estaciones con su respectiva propiedad de distancia, creando en una combinación en 2 de 622 estaciones 193131 aristas `S_DISTANCE` entre las distintas `(:Station)`, como se muestra en la figura 6.

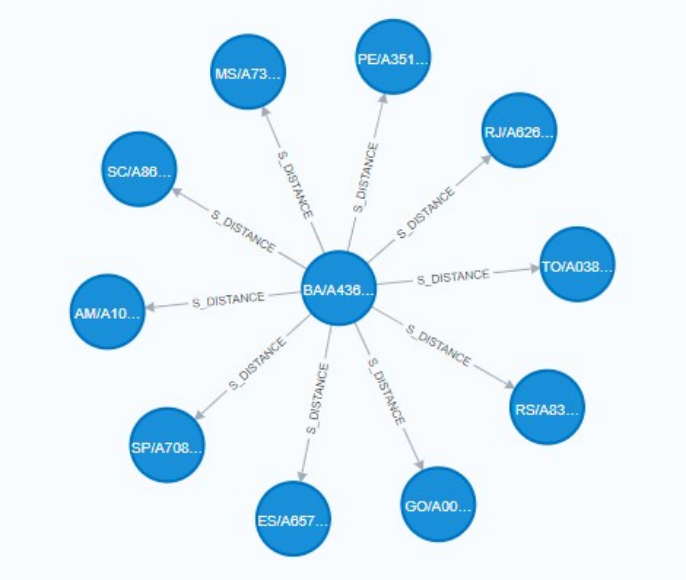

Figura 6: Nodos `(:Station)` y algunas de sus aristas `S_DISTANCE`

Habiendo modelado el subgrafo espacial con aristas con la propiedad de distancia, la consulta que se quiere resolver en el apartado anterior resulta sencilla de formular:

```
MATCH (m0:Measurement)<--(s0:Station{name: "MS/A756/AGUA CLARA"})
-[d:S_DISTANCE]-(s:Station)-->(m:Measurement{hour: m0.hour, date: m0.date})
WHERE m0.date = date("2019-01-01") AND m0.hour.hour = 2 and d.distance < 300
RETURN s0, s
```

Para esta consulta el tiempo de ejecución es del orden de los 20ms (cuadro III), comprobando efectivamente que la consulta es mucho más rápida.

A modo de comparación con el apartado anterior donde se evalúan las distintas consultas utilizando la función `distance()` de *Cypher*, podemos decir que el hecho de modelar las aristas con el dato de distancia entre las estaciones evita la necesidad de utilizar dicha función en tiempo de consulta, que como se comprobó, tiene un costo computacional considerable.

II-D. Algunas consultas de ejemplo

En esta sección se presentan algunos casos de uso que parecen interesantes, y podrían ser relevantes en aplicaciones reales de meteorología, y principalmente, que utilizan como base las consultas que hemos ido desarrollando a lo largo del proyecto.

Es importante aclarar que la utilidad en sí de estas consultas particulares no es el foco del proyecto, sino que simplemente se muestran como ejemplos ilustrativos para entender en qué medida podría ser importante tener formas eficientes de acceder a los registros utilizando consultas similares a las anteriores.

Cuadro III: Tiempos de ejecución promedio de las consultas espaciales sin índices

Consulta	**Tiempo promedio de ejecución (n=5)**	**Desviación estándar (n=5)**
Estaciones y mediciones con cálculo de distancia posterior	98.4652s	0.8201s (0.83 %)
Filtrando primero por distancia entre estaciones	6.1026s	0.6500s (10.65 %)
Filtrando primero por mediciones y luego por distancia	0.0580s	0.0.468s (80.66 %)
Subgrafo espacial (aristas con propiedad de distancia)	0.0186s	0.0027s (14.30 %)

Uno de los problemas que se encuentra habitualmente en meteorología, es la falta de datos de algunas estaciones para algunos intervalos de tiempo, o la interpolación de las medidas ya sea temporal o espacialmente, para puntos que no coinciden exactamente con las mediciones (e.j: un punto entre dos o más estaciones, o a una hora no puntual).

Aquí se usará la variable temperatura solamente a modo de ejemplo, y las particularidades que se deberían tener en cuenta para esta u otras variables están por fuera del alcance de este trabajo. El interés en este caso es la consulta de los datos relevantes de la base, y no el modelo exacto de predicción a utilizar, por lo tanto simplemente usaremos promedios con datos relacionados.

Suponiendo que se quiere estimar un dato faltante para cierta estación, se podría hacer un promedio de las estaciones cercanas (sin ponderar por distancia, aunque sería posible hacerlo sin cambiar el tiempo de ejecución de la consulta), en el mismo horario que se quiere estimar. Además del valor estimado, es bueno tener una idea del error en la predicción, y para ello se puede repetir el procedimiento para días en los que sí había datos para la estación, y observar la diferencia.

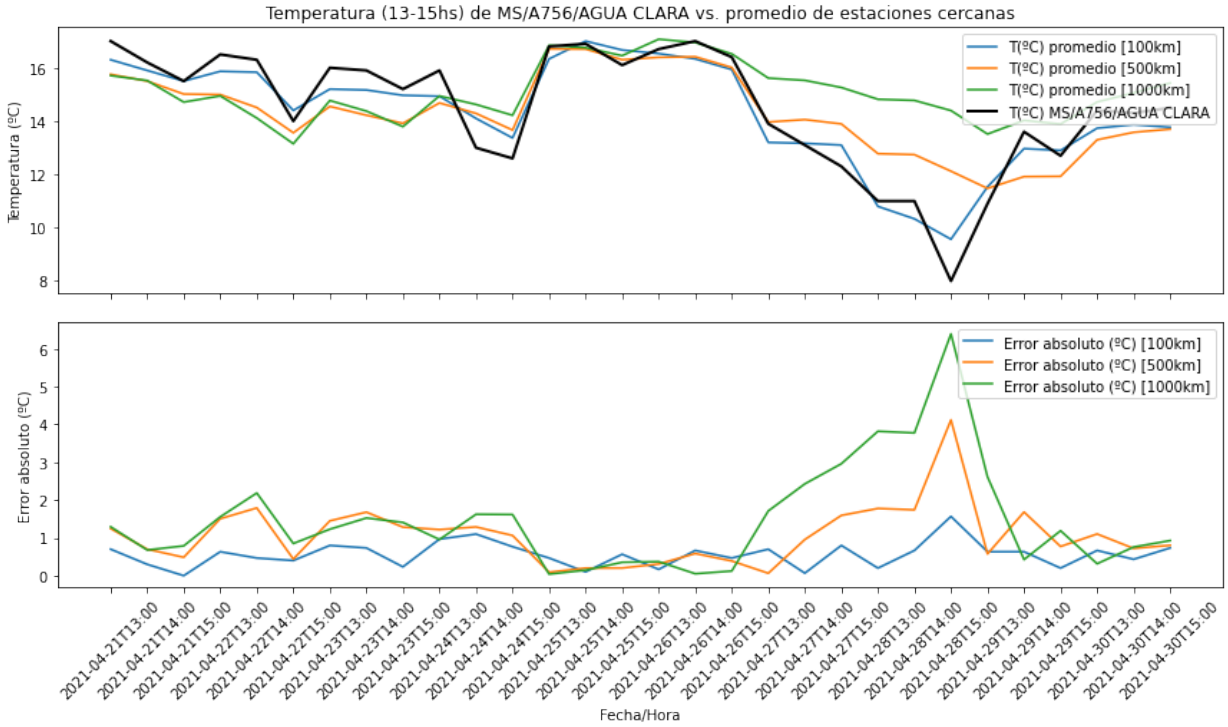

Figura 7: Predicción de temperatura promedio para cierto momento del día (13, 14 y 15 horas) basado en mediciones de días anteriores, por estaciones con distinta cercanía. Se calcula el error con la medición real de la estación (que no se incluye en el promedio), para estimar cual es el rango de distancia a considerar, que arroja mejores estimaciones.

En la figura 7 se puede ver una consulta que estima no sólo el valor para una estación, sino el error para varios momentos anteriores similares, agrupados además por el radio de distancia que se toma para buscar las estaciones cercanas. En este caso, se observa que usando un radio de 100km se logran mejores estimaciones que tomando distancias mayores, en la gran mayoría de los puntos evaluados.

Estos hallazgos son posibles gracias a que el tiempo de ejecución de las consultas espaciales y temporales son muy cortos, y se pueden ejecutar repetidas veces (en este ejemplo, para distintos radios de búsqueda) desde la capa de aplicación según sea necesario, para alimentar modelos más complejos que no pueden ser codificados en un lenguaje de consultas como Cypher.

En la figura 8 se puede ver otro ejemplo interesante, que es estimar el error fijando el día y hora, pero variando la estación. Esto se hace repitiendo una consulta muy similar a la anterior, pero para cada estación cercana a la estación de interés (marcada con un punto negro como `AGUA CLARA`).

Se puede observar que la estación de interés está en una zona donde las predicciones parecen ser buenas, en comparación con lo que sucede por ejemplo en la zona cercana de `ALTO TAQUARI`, donde probablemente alguna particularidad geográfica genera condiciones meteorológicas diferentes a la región. En este caso se usó un radio de 500km alrededor de la estación de interés para encontrar las estaciones a mostrar, y un radio también de 500km alrededor de cada estación para calcular su propio error de predicción (es decir, hay estaciones que no se muestran, que entran en el cálculo de error de las estaciones de los bordes). Este caso de uso es interesante porque hace uso extensivo de la distancia entre estaciones, y por lo tanto requiere tener consultas eficientes en ese sentido para realizarse en un tiempo razonable. En este caso, todas las consultas necesarias para generar el gráfico se ejecutan en muy pocos segundos.

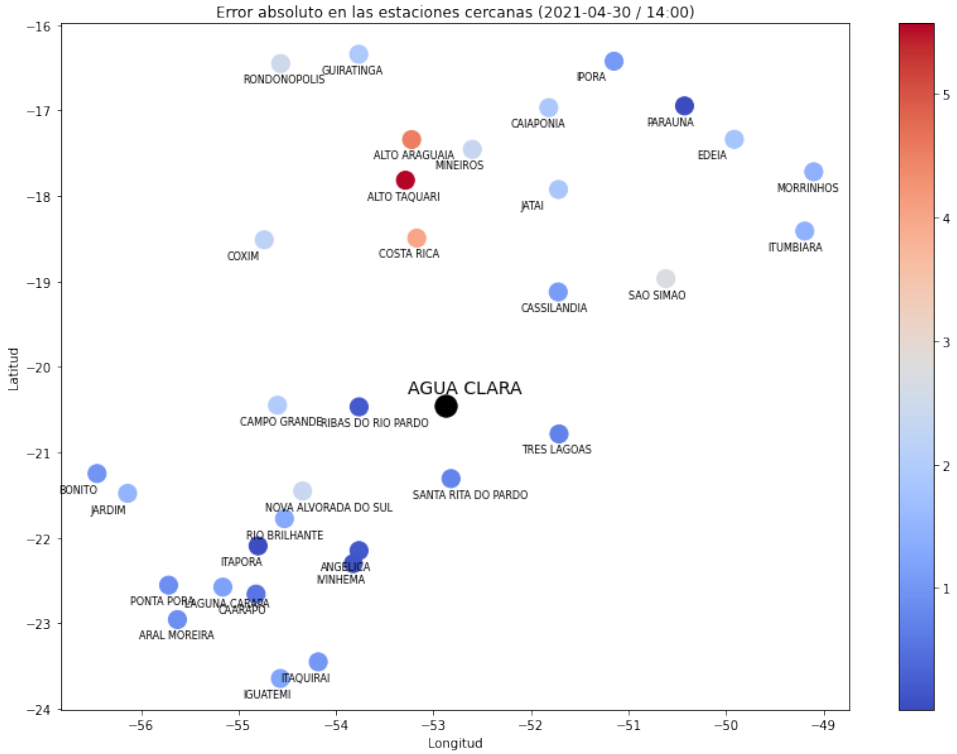

Figura 8: Error en las predicciones para distintas estaciones alrededor de una estación de interés (radio de 500km), que permite encontrar zonas con temperaturas regulares superficialmente (puntos más azules) o zonas con particularidades meteorológicas (puntos rojos) que tienen mediciones distintas al promedio de las estaciones cercanas.

Todo el código necesario para generar la base de datos y correr estas consultas o comparar los resultados en diferente hardware se puede encontrar en el código del proyecto [4].

III. Conclusiones

A lo largo del proyecto se pudieron comparar distintas soluciones al problema de la búsqueda por atributos temporales y espaciales. En la búsqueda temporal se pudo observar que generar nodos intermedios con los valores de los atributos, para generar subgrafos más reducidos y restringir el espacio de búsqueda, puede ayudar en general, aunque no en todos los casos. Esto debe estar complementado y alineado con los índices sobre los atributos relevantes cuando sea posible, ya que éstos evitan la necesidad de recorrer todos los registros y potencialmente pueden acelerar algunas consultas en varios órdenes de magnitud. Por ejemplo, en el caso del primer subgrafo temporal, no se observó una mejora en las consultas al crear los índices, aunque sí con el segundo subgrafo temporal donde se usa un nodo para cada día y hora, y puede hacer uso de un índice pero genera por otro lado una sobrecarga relevante (dependiente del intervalo de tiempo y resolución necesarias) en el número de nodos.

En la búsqueda espacial se observó que el cálculo de una función relativamente simple como la de distancia, puede generar consultas muy lentas si se deben evaluar muchas combinaciones de registros, y por lo tanto puede ser una buena opción realizar el cálculo una sóla vez y crear aristas entre los registros con los valores precalculados.

Finalmente, se mostraron algunos casos de uso que son factibles gracias a que las consultas tanto temporales como espaciales son muy rápidas, ya que de otra manera sería necesario esperar varios minutos para obtener la misma información.

Aunque algunas de estas técnicas pueden ser extensibles a las bases de datos relacionales, la ventaja de una base de grafos está en que las aristas que salen de cada nodo son accesibles de forma muy rápida sin necesidad de buscar en una tabla de relaciones *n:m*, y esto permite que algunas consultas que implican filtrar por relaciones entre nodos sean más rápidas además de ser más intuitivas. Por ejemplo, para precalcular las distancias, la solución encontrada es más intuitiva y probablemente más rápida que lo que se podría lograr con una tabla relacional, y lo mismo sucede con la relación entre nodos de día y hora y los de medidas.

Referencias

[1] How-To: Import CSV Data with Neo4j Desktop - Developer Guides.

[2] Spatial functions - Neo4j Cypher Manual.
[3] Batch transactions with CALL clause - Neo4j Cypher Manual, 2022.
[4] Código del proyecto, July 2022.
[5] Bin Feng, Qing Zhu, Mingwei Liu, Yun Li, Junxiao Zhang, Xiao Fu, Yan Zhou, Maosu Li, Huagui He, and Weijun Yang. An Efficient Graph-Based Spatio-Temporal Indexing Method for Task-Oriented Multi-Modal Scene Data Organization. *ISPRS International Journal of Geo-Information*, 7(9):371, September 2018.
[6] Dave Gordon. Warm the cache to improve performance from cold start - Knowledge Base.
[7] INMET - Brasil. Climate Weather Surface of Brazil - Hourly, May 2021.
[8] Craig Taverner. Building Spatial Search Algorithms for Neo4j, September 2020.
[9] Dan Williams. Graphs in Time and Space: A Visual Example, October 2018.
[10] Michael Zelenetz. Graphing Space and Time, December 2019.