Estudio de persistencia de datos del Sistema de Transporte Metropolitano

Ramiro Nieto *Estudiante de Grado*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
ramiro.nieto@fing.edu.uy
Santiago Correa *Estudiante de Grado*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
santiago.correa.perini@fing.edu.uy

Resumen

El trabajo de investigación descrito en este informe busca realizar un análisis comparativo de la resolución de un problema en dos motores de bases de datos no relaciones distintos. El problema planteado es el siguiente: Dadas dos paradas A y B por las cuales pasa más de una línea, buscamos saber cual es la línea menos concurrida teniendo en cuenta si el día es hábil o no y dada una franja horaria determinada.

I. Introducción

Este trabajo esta motivado en la posibilidad de investigar y hacer uso de los datos abiertos de Uruguay. Luego de un análisis de los mismos, se concluye que los más consistentes y actualizados son los provistos por la Intendencia de Montevideo (IM). Dentro de los conjuntos de datos, surgió un interés particular en los datos relacionados al transporte de ómnibus, ya que se observó potencial para un modelado fácil y consecuente uso de las herramientas asociadas a una estructura de grafo.

Debido a adversidades encontradas en el posterior análisis, no solamente se modelaron mediante una base de datos de grafos, si no que se modelo en una base de datos documental para realizar un análisis comparativo entre las dos implementaciones.

En un principio, la implementación en una base de datos de grafos se elige, además de su asociación trivial, por la capacidad de poder aplicar algoritmos como el de camino mas corto ponderado. Para esto se utilizó Neo4j AuraDB debido a la documentación de la misma y por ser una opción gratuita.

Por el lado de la implementación en una base de datos documental, se decidió usar MongoDB por motivos de anterior experiencia con la sintaxis y motor.

II. Trabajos relacionados

No se encontraron trabajos similares ya que, a diferencia de otros trabajos que usan bases de datos de grafos, en este caso el camino es fijo. Debido a esto, no se necesita usar una función que determine el camino más corto. Por el lado de la base documental tampoco se encontraron trabajos suficientemente relevantes como para ser citados. La especificidad y originalidad de la propuesta nos limito en este sentido.

Un concepto a tener en cuenta a lo largo del informe es el de variante de una linea. La variante de una línea es lo que coloquialmente conocemos como 'el de ida' y 'el de vuelta'. Pero a su vez, una línea podría tener una variante distinta debido a que su parada destino es una anterior en el recorrido original. Por lo tanto, no solamente se diferencia entre lineas, si no que por línea y su variante.

III. Alcance de la motivación y trabajo realizado

III-A. Análisis inicial

Esta sección consistirá en describir como se llegó a definir un alcance del proyecto, a partir de la motivación descrita en la introducción, y el trabajo realizado acorde. Esto incluye las decisiones tomadas y adaptaciones necesarias de acuerdo al análisis y diseño de la problemática. La motivación inicial es simple; frente a un planteo conocido donde su modelado en grafos es intuitivo, usar algoritmos conocidos y optimizados para aprovechar el hecho de persistir los datos obtenidos en Neo4j. Rápidamente, al comenzar el trabajo, vimos que esta aproximación por más que es válida, no la consideramos como óptima.

El modelado consiste en un conjunto de nodos que representan las paradas del Sistema de Transporte Metropolitano (STM) y un conjunto de aristas para indicar la relación entre dos paradas A y B. Para toda línea de ómnibus y variante del STM que vaya de A a B en su recorrido, se forma una relación entre A y B con la cantidad de boletos emitidos en A para dicha línea y variante. Esto involucra tener en cuenta el día y horario de la emisión, por lo tanto habrán tantas relaciones como franjas horarias y día. A efectos de generar un primer acercamiento a la problemática se define la siguiente categorización para las emisiones:

- Se divide el día en 8 franjas horarias a tener en cuenta sobre que horario se realizó la emisión.
- Si se emitió en un día hábil o no

Teniendo en cuenta el modelado, la traducción a realizar para poder hacer uso de algoritmos como el camino mas corto ponderado fue inmediata. Los datos provistos por la IM [de Movilidad IMM, s.f.-a], permiten realizar este modelado realidad. Por mes, están los boletos emitidos con la información necesaria asociada y se puede inferir el recorrido de cada línea a través de la descripción de las paradas. Referirse al anexo VI para la explicación del procesamiento de datos y acceso al código asociado.

En la base de datos de grafos se hizo el siguiente modelado de datos [1]. Se crean nodos que representan paradas, y relaciones entre ellas representando las líneas y variantes, donde el nombre de la relación tiene la información del número y variante de la línea, si el día es hábil o no y la franja horaria. El string que representa el nombre tiene el siguiente formato numero-variante-bit hábil o no-franja horaria. El numero de línea es un string ya que hay líneas como el CE-1. La variante también es un string ya que hay variantes con letras. Además, hay una relación por cada franja horaria y día hábil o no, ya que el hecho de que una línea esté más congestionada puede depender del horario y día. Esto, sin embargo, causa que hayan muchas relaciones entre dos nodos debido a que habrá una relación por cada línea, franja horaria y día hábil o no. Por ejemplo, si entre dos paradas sólo pasa una línea tenemos al menos 16 relaciones entre esos dos nodos, 8 por cada franja horaria de 3hs y esto por dos para considerar si el día es hábil o no. Este formato fue elegido porque se pensó que reducía el tiempo al correr la consulta y también, simplifica la consulta que responde la pregunta planteada, a cambio de tener muchas relaciones en la base.

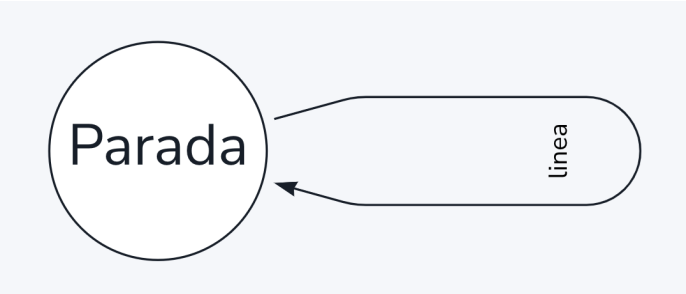

Figura 1: Modelado de la base de datos de grafos.

III-B. Cambio de enfoque

El problema respecto a la realidad planteada de esta manera es que realizar un algoritmo sobre el grafo para obtener y comparar caminos, como lo es el camino mas corto ponderado, carece de sentido. Los recorridos de cada línea ya determinan los caminos al momento de tomar una decisión de que relación tener en cuenta entre dos nodos parada. Por lo tanto, las consultas a realizar sobre la base no aprovechan el tipo de base donde se persisten los datos. El enfoque del trabajo se adaptó con el fin de comparar la eficiencia de las consultas entre una base de datos documental (MongoDB) y una de grafos (Neo4j), y las diferencias en utilidad que puedan surgir.

Debido a este cambio, se incorporaron las decisiones a tomar respecto al análisis y diseño de los datos para persistirlos en una base de datos documental. De acuerdo a lo estudiado en el curso, para tomar las decisiones de diseño respecto a las colecciones y documentos de la base de datos, nos basamos en priorizar la estructura de acuerdo a la consulta. Teniendo en cuenta la especificidad de los datos, se resolvió incluir una sola colección con documentos que siguen la siguiente estructura:

Listado 1 Estructura de un documento de la colección

```
{ 'id': string,
  'numero': string,
  'paradas_flujo_habiles': array,
  'paradas_flujo_inhabiles': array,
}
```

Donde:

- **id** es el id de la linea.
- **numero** es la concatenación del número público de la linea, un guión, y el número de variante de la linea.
- **paradas_flujo_habiles** es un arreglo de largo igual a la cantidad de paradas que tiene el recorrido de la linea. Cada celda contiene un arreglo de largo 8, donde se guardan la cantidad de boletos emitidos en cualquier día hábil, por franja horaria.
- **paradas_flujo_inhabiles** contiene lo mismo que la propiedad anterior, solamente que los boletos emitidos son en un día no hábil.

De esta manera, las consultas a realizar solamente involucran agregar en la agregación dos *unwind* para tener todos los datos correspondientes a una línea y variante dada. Igualmente, como se menciona en el trabajo futuro a tener en cuenta [V], esta decisión de diseño puede cambiar en caso de implementar más información a tener en cuenta.

III-C. Falta de heurística y fidelidad

Con el objetivo principal de saber cual es la línea menos concurrida, hay un dato que falta esencial: cuántas personas se bajan por parada. Este dato es clave y es uno que la IM ha hecho trabajo para conseguir. Logramos comunicarnos con el Departamento de Desarrollo Sostenible e Inteligente, donde nos comunicaron que "*la gente del STM están en un plan piloto que incluye hardware a bordo etc, para tener ese dato mas fiel*".

Esto hace que todo el análisis se haga solamente sobre la gente se se sube al ómnibus en vez de la cantidad de gente arriba del ómnibus en un momento dado. Este razonamiento tiene claras fallas. Quizás en una línea entre dos paradas dadas, se sube más gente en una que otra, pero al mismo tiempo se baja más gente, haciendo que termine estando más vacío a pesar de haberse subido más gente.

Sin embargo, la forma en la que se realizó la resolución al problema permite sustituir la cantidad de gente que se sube con una estimación de la cantidad de gente que sube menos la que se baja por parada, obteniendo un resultado más cercano a la realidad en caso de tener disponible la heurística que estime la gente a bordo o el dato concreto.

IV. Experimentación

Para la experimentación en ambas implementaciones, se usaron los datos de los viajes de Abril 2022 [de Movilidad IMM, s.f.-c] y los datos de los recorridos 2022 [de Movilidad IMM, s.f.-b]. La consulta de principal interés es la siguiente:

Dadas dos lineas y sus variantes, una parada origen y destino en la que coincidan en sus respectivos recorridos, indicar en cual de los dos ómnibus es mas probable que se suba menos gente en un día hábil en cierta franja horaria

IV-A. En MongoDB

Para la implementación en MongoDB, se realizó la consulta deseada y dos adicionales. La consulta deseada es la siguiente:

Listado 2 Primera consulta MongoDB

```
pipeline = [
            {'$match': {
               'numero': {'$regex': f'({linea_var_1}|{linea_var_2})'},
            }},
            {
             '$unwind': {'path': '$paradas_flujo_habiles', 'includeArrayIndex': 'ordinal'
            }},
            {'$match' : {'$or': [
                           {'$and': [
                              {'numero': linea_var_1},
                              {'$expr' : {'$gt' : ['$ordinal', ordinal_origen_1 - 1]}},
                              {'$expr' : {'$lt' : ['$ordinal', ordinal_destino_1 - 1]}}]},
                           {'$and': [
                              {'numero': linea_var_2},
                              {'$expr' : {'$gt' : ['$ordinal', ordinal_origen_2 - 1]}},
                              {'$expr' : {'$lt' : ['$ordinal', ordinal_destino_2 - 1]}}]}
                           ]
                     }
            },
            {
             '$unwind': {'path': '$paradas_flujo_habiles', 'includeArrayIndex': 'index'
            }},
            {
             '$group': {
               '_id': {'index': '$index', 'linea_var' : '$numero'},
               'cantidadBoletos': {'$sum': '$paradas_flujo_habiles'}
             }
            },
            {
               '$sort': {
                  '_id.index': -1
               }
            }
         ]
```

Como se puede observar, se necesita además de saber las lineas y sus variantes, el ordinal de las paradas en cada recorrido. La consulta consiste en primero filtrar por las dos lineas deseadas, y a través de un primer *unwind* del arreglo **paradas_flujo_habiles**, se obtienen los datos correspondientes según parada debido a como se diseño y procesaron los datos.

El siguiente paso es filtrar según los ordinales correspondientes para cada línea, de manera tal que solamente queden las paradas entre medio del origen y destino. Una vez conseguido esto, se tiene el siguiente *unwind* para poder diferenciar por franja horaria. Se concluye la consulta agrupando según línea y franja.

Como ejemplo, tomamos el caso de la línea 300 y 407, con parada origen ubicada en Minas y Av Gonzalo Ramírez, y parada destino Av Italia y Ricaldoni. En caso de tomar la franja horaria de quince horas a dieciocho horas, teniendo en cuenta los días hábiles:

Listado 3 Respuesta primera consulta MongoDB

```
Para la parada en AV GONZALO RAMIREZ esquina MINAS
hasta la parada en AV ITALIA esquina RICALDONI
en los dias habiles de Abril 2022 sumados


En la franja horaria 15-17
En la linea y variante 300-8385 1609 personas se subieron
En la linea y variante 407-49 463 personas se subieron


Demoro en hacerse la consulta: 0.046588 segundos.
```

A partir de la respuesta se puede concluir que en este caso conviene tomarse el 407.

A modo de evaluar otras posibilidades, se realizaron las siguientes consultas. La segunda consulta responde, dada una línea y su variante, la cantidad de boletos emitidos por franja horaria en todo su recorrido.

Listado 4 Segunda consulta MongoDB

```
pipeline = [
            {'$match': {
               'numero': linea_var,
            }},
            {
             '$unwind': {'path': '$paradas_flujo_habiles'
            }},
            {
             '$unwind': {'path': '$paradas_flujo_habiles', 'includeArrayIndex': 'index'
            }},
            {
             '$group': {
               '_id': '$index',
               'cantidadBoletos': {'$sum': '$paradas_flujo_habiles'}
             }
            },
            {
               '$sort': {
                  'cantidadBoletos': -1
               }
            }
         ]
```

Su respuesta para el caso de la línea 60 y variante 2393 (Portones-Ciudadela) es la siguiente:

Listado 5 Respuesta segunda consulta MongoDB

```
Para la linea y variante 60-2393


En la franja horaria: 12-14 se emitieron 3953 boletos.
En la franja horaria: 9-11 se emitieron 3659 boletos.
En la franja horaria: 15-17 se emitieron 2818 boletos.
En la franja horaria: 06-08 se emitieron 1987 boletos.
En la franja horaria: 18-20 se emitieron 1767 boletos.
En la franja horaria: 21-00 se emitieron 402 boletos.
En la franja horaria: 00-02 se emitieron 109 boletos.
En la franja horaria: 03-05 se emitieron 2 boletos.


Demoro en hacerse la consulta: 1.495767 segundos.
```

Y por último, dadas dos lineas y sus variantes, donde coincidan en una parada en su recorrido, indicar para una franja horaria, cuantos boletos se emitieron antes de llegar a la parada.

Listado 6 Tercera consulta MongoDB

```
pipeline = [
            {'$match': {
               'numero': {'$regex': f'({linea_var_1}|{linea_var_2})'},
            }},
            {
             '$unwind': {'path': '$paradas_flujo_habiles', 'includeArrayIndex': 'ordinal'
            }},
            {'$match' : {'$or': [
                           {'$and': [
                              {'numero': linea_var_1},
                              {'$expr' : {'$lt' : ['$ordinal', ordinal_linea_1]}}]},
                           {'$and': [
                              {'numero': linea_var_2},
                              {'$expr' : {'$lt' : ['$ordinal', ordinal_linea_2]}}]}
                           ]
                     }
            },
            {
             '$unwind': {'path': '$paradas_flujo_habiles', 'includeArrayIndex': 'index'
            }},
            {
             '$group': {
               '_id': {'index': '$index', 'linea_var' : '$numero'},
               'cantidadBoletos': {'$sum': '$paradas_flujo_habiles'}
             }
```

```
        },
        {
            '$sort': {
                '_id.index': -1
            }
        }
    ]
```

Para la parada ubicada en Julio Herrera y Reisig esquina Giribaldi, comparando entre el 300 y 407 para la franja horaria entre quince horas y dieciocho horas, se obtiene lo siguiente:

Listado 7 Respuesta tercera consulta MongoDB

```
Para la parada en AV TOMAS GIRIBALDI esquina AV JULIO HERRERA Y REISSIG
en los dias habiles de Abril 2022 sumados


En la franja horaria 15-17
La linea y variante 300-8385 tuvo 801 boletos comprados antes
La linea y variante 407-49 tuvo 547 boletos comprados antes


Demoro en hacerse la consulta: 0.048111 segundos.
```

IV-B. En Neo4j

Para obtener la respuesta a la pregunta planteada sobre la base de datos de grafos se realizaron las siguientes consultas

Listado 8 Primera consulta en CYPHER

```
1  MATCH p = (p1:Parada {parada: '4418'})-[r:`407-49_1_5`*]-(p2:Parada {parada: '2122'})
2  RETURN reduce(s=0, i in relationships(p) | s+apoc.convert.toInteger(i.personas))
```

Listado 9 Segunda consulta en CYPHER

```
1  MATCH p = (p1:Parada {parada: '4418'})-[r:`300-8385_1_5`*]->(p2:Parada {parada: '2122'})
2  RETURN reduce(s=0, i in relationships(p) | s+apoc.convert.toInteger(i.personas))
```

Estas consultas buscan retornar la cantidad de gente que se subió entre la parada 4418 y se bajaron en la parada 2122. El 1 de la etiqueta de la relación indica que el día es hábil y el 5 indica que es la 5ta franja horaria, de 15 a 18 horas. Se diferencian únicamente en la línea y variante escogidas.

Se usa la función reduce con el objetivo de sumar todas las personas que se subieron al ómnibus entre las paradas. Por otro lado se usa la biblioteca apoc para convertir en integer la cantidad de personas de la propiedad de la relación.

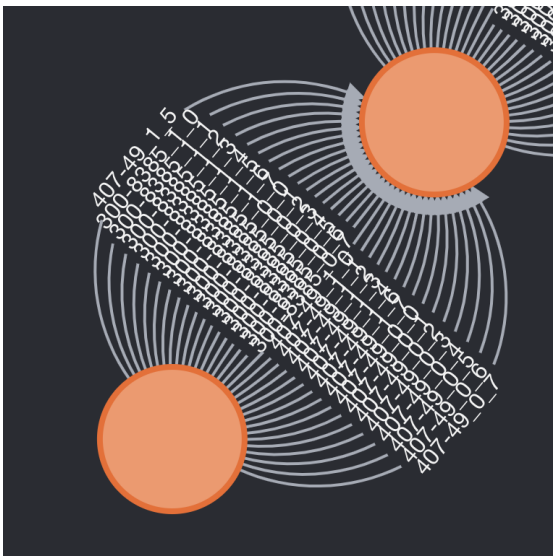

Figura 2: Múltiples aristas entre grafos.

V. Conclusiones y trabajo futuro

Con respecto al objetivo planteado inicialmente, se llega a responder la pregunta de cuantas personas se suben entre dos paradas dada una franja horaria y si el día es hábil o no. Esta pregunta se puede hacer para dos lineas distintas obteniendo en cual de las líneas se sube menos gente entre dos paradas en las que coincidan. Esto es uno de los posibles indicadores de que esta línea está menos congestionada. Se obtienen otros indicadores posibles que se pueden analizar con los datos disponibles en las bases, como la cantidad de gente que se subió hasta que se llega a la parada origen.

Como se estableció en [III], hay una gran oportunidad de realizar las mismas consultas pero con la heurística, indicando cuantas personas se encuentran en el ómnibus entre dos paradas dada una franja horaria y si el día es hábil o no. Con este dato se pueden hacer otras consultas, como elegir cual es la línea que tiene menor cantidad máxima de pasajeros o cual es la que tiene media más baja de pasajeros. Sobre todo, tiene valor a la hora de poder dar datos más prácticos, estando más cerca de solucionar el problema planteado originalmente.

Para corroborar el buen funcionamiento de las bases de datos, se usó la línea 407 variante 49 y 300 variante 8385, que ambas coinciden desde la parada con id 4418 a la con id 2122. Para la consulta en MongoDB se obtienen los ordinales de cada parada en el recorrido de cada linea, y en Neo4j se realiza la consulta desde la siguiente parada (id 4418) ya que no se tiene en cuenta las personas que se suben en la para origen. Se tomo la franja horaria de quince a dieciocho en días hábiles, y se llegó a que ambas consultas en las diferentes bases de datos tuvieron el mismo resultado. Esto concluye que en Abril se subieron 463 personas a la línea 407 y 1609 al 300.

Dados los ajustes a la pregunta inicial mencionados anteriormente, se procedió a comparar al eficiencia de las consultas entre las dos bases de datos. Las consultas en Neo4j se realizaron en la herramienta *Neo4j Browser*. La primera vez que se ejecutó una de las consultas se obtuvo que el tiempo demorado fue 225ms. Sin embargo, una vez corrida esta primera consulta, las otras demoraron solamente 1ms, incluso cambiando la línea y variante de la consulta. Por el lado de MongoDB, los tiempos entre que se ejecuto la primera consulta y se obtuvo del lado del usuario fueron 0.04 1.10 y 0.04 segundos para las respectivas consultas especificadas en [IV]. Como trabajo futuro consideramos que se puede hacer un mejor análisis de los tiempos de las consultas.

Por otro lado, un posible trabajo a futuro es agregar esta funcionalidad a una aplicación que tenga los tiempos de llegada de una línea a una parada donde el usuario esté (ya que en Montevideo muchas líneas cuentan con GPS), con el objetivo de poder informar a un usuario. En el caso de que hayan dos lineas que llegan relativamente al mismo tiempo, el usuario podrá optar por esperar a una de las dos líneas y subirse a la que este menos congestionada. Esto aplicado en un grupo de usuarios grande y con datos actualizados, podría llegar a mejorar la congestión de ciertas líneas, la experiencia del usuario con el STM o a justificar el agregado de ómnibus a líneas congestionadas.

A su vez, otra oportunidad de mejora surge respecto a los datos subidos a la base de datos de grafos. No se pudo subir todos los datos de las relaciones de la forma que fueron modeladas, debido a que la versión gratis de Neo4j tiene un límite de tamaño que pueden tener los datos almacenados (de 250MB), el cual es excedido al intentar se subir todas las relaciones. Se subieron las primeras 100 mil relaciones y luego las necesarias para hacer la comparación de las consultas hechas en Mongo. Relacionado a este tema también se pudo haber incluido más información en los nodos parada, como los nombre de las calles en las que están o permitir mas especificidad en las franjas horarias (más chicas) y en los días (por cada día de la semana).

Y por último, por el lado de la implementación en una base de datos documental, se podría ver como oportunidad de mejora, incluir otra colección referida a las paradas. De esta manera, se evita el manejo de tantos parámetros de entrada para realizar una consulta y enriquece las respuestas con detalles sobre las paradas, pensando en el uso por parte de un usuario.

VI. Anexo

En esta sección se tiene en cuenta los conjuntos de datos mencionados anteriormente de viajes en Abril 2022 y recorridos por línea de 2022.

El primer conjunto consiste en un archivo *.csv* donde cada línea representa un boleto emitido. La información de interés asociada al boleto es la linea, su variante, la parada y la fecha en la que se emitió. El segundo conjunto consiste en 4 archivos que conforman los recorridos. A través de la librería de *Python*, *pyshp*, se pueden leer los 4 archivos generando un marco de datos donde cada fila contiene (de interés al proyecto) la linea, su variante, la parada asociada a la línea y su ordinal en la recorrida de la variante. De la misma manera, usando la librería *pandas*, se genera un marco de datos para los boletos.

Antes de comenzar el procesamiento de datos, se realizó la separación del conjunto de datos con los boletos emitidos, ya que es un archivo de casi tres giga bytes. Se formaron 9 archivos separados de dos millones seiscientas mil lineas cada uno, excepto el noveno y último archivo. También en el proceso de generar los datos para la base de datos en MongoDB, encontramos inconsistencias entre los dos conjuntos. Hay boletos emitidos para variantes de lineas que no tienen un recorrido asociado o en paradas que no se encuentran en el recorrido. Para el primer caso, no encontramos una razón posible para justificar la diferencia. Sobre la segunda inconsistencia, se podría explicar por el registro de un desvío particular. De todas maneras, como se observo que ocurría en la minoría de los casos, se resolvió solamente tener en cuenta los datos consistentes entre los dos conjuntos e ignorar el resto.

Todos los archivos generados mencionados en esta sección se encuentran en un repositorio de GitLab[1].

VI-A. JSONs para MongoDB

De acuerdo a las decisiones de diseño tomadas anteriormente, primero se necesitan tener los recorridos para poder indicar correctamente en el arreglo de paradas del documento, a que celda del arreglo pertenece el boleto.

Se busca generar un conjunto de *JSONs* con el siguiente formato:

Listado 10 Estructura de un JSON del archivo de recorridos

```
{ 'numero': string,
  'paradas': {'parada': string},
}
```

donde:

- En la propiedad *numero* se guarda la línea y su variante, separada por un guión
- En la propiedad *paradas* se guarda por cada *parada* su ordinal en el recorrido correspondiente. Cada parada tiene su identificador global único.

Recorriendo el conjunto de datos, por cada fila del marco de datos, se ingresa la parada en su línea y variante correspondiente, teniendo en cuenta su ordinal.

Una vez obtenido este archivo (*recorridos_por_linea.json*), se puede realizar el procesamiento del segundo conjunto de datos. Recorriendo cada fila, se genera un *JSON* con el formato deseado [III-B], si es que ya no se creo, para cada línea y variante. Verificando la consistencia con el recorrido de los datos del boleto, se extrae el ordinal correspondiente a la parada para dicha línea y variante. De esta manera, se suma en uno en el índice de franja correspondiente según la fecha y hora, para la celda correspondiente del arreglo hábil o no hábil que representa a dicha parada del recorrido. Es decir que si, para la línea 60 variante 2393 se encuentra un boleto en un día hábil para la franja horaria entre quince horas y dieciocho horas y esa parada corresponde a la novena en el recorrido (ordinal nueve), entonces en el documento de dicha línea y variante, para la propiedad *paradas_flujo_habiles* en la octava celda, se le suma en 1 a la quinta celda del arreglo contenido dentro (que corresponde a la franja mencionada).

De esta manera se obtiene el archivo *tickets_abril.json* que contiene para cada línea y variante, su correspondiente documento de la colección con los datos de los boletos emitidos según parada y franja cargados.

VI-B. CSVs para Neo4j

Al empezar primero por generar los datos en formato *JSON* para ser subidos a Mongo, estos archivos fueron usados para construir los archivos *.csv* que fueron subidos a Neo4j, ya que se considero más practico que intentar subir los archivos *JSON* con los cambios necesarios. Neo4j tiene una herramienta llamada *Data Importer*, la cual se usó para el cargado de los nodos. Para el mismo, se creo un primer archivo *.csv* que donde cada fila tiene el número de la parada. Luego, con la herramienta nombrada, la creación de los nodos parada fue trivial. Para las relaciones se tuvo que crear un segundo archivo *.csv* que consta

[1] https://gitlab.fing.edu.uy/santiago.correa.perini/proyecto-final-bdnr-2022

de 4 columnas: el nombre generado mediante código *Python*, la parada salida, la parada destino y la cantidad de gente que se subió en ese tramo. Para cargar las relaciones, como el nombre de la misma estaba dentro del archivo y no todas las entradas del *.csv* tienen el mismo nombre, se tuvo que usar la biblioteca *apoc* de Neo4j que provee una función que permite crear una relación Neo4j, s.f. sin precondiciones, pudiendo poner el nombre buscado a cada relación. Este archivo constaba de alrededor de 350mil entradas, siendo cada entrada una relación distinta.

Referencias

de Movilidad IMM, D. (s.f.-a). *Datos abiertos de Transporte*. https://catalogodatos.gub.uy/organization/d9026405-35be-410e-b251-492fba99c57d?organization=intendencia-montevideo&groups=transporte

de Movilidad IMM, D. (s.f.-b). *Recorridos por linea y variante 2022*. https://intgis.montevideo.gub.uy/sit/php/common/datos/generar_zip2.php?nom_tab=v_uptu_paradas&tipo=gis

de Movilidad IMM, D. (s.f.-c). *Viajes de Abril 2022*. https://imnube.montevideo.gub.uy/share/s/THiuXt2vRsCLypFeNaMoIg

Neo4j. (s.f.). *Create relationship*. https://neo4j.com/labs/apoc/4.4/overview/apoc.create/apoc.create.relationship/