Project : Try to reproduce CityMapper(transportation App) results in Paris using Neo4J Graph

In this project, I will only try to reproduce CityMapper's results for a few days ahead (real time results depend on more factors that come directly from an API).

I am using Neo4J graph database, in order to calculate as quickly as possible the shortest path using Dijkstra's algorithm.

1st step : download data from open data paris

Data are stored following Google's GTFS format.
For paris data, we can download them here :
https://data.ratp.fr/explore/dataset/offre-transport-de-la-ratp-format-gtfs/information/

2nd step : Data preprocessing (see city mapper.iypnb)

The objective here is to calculate the duration between two train stations, for that we will use pandas shift function and remove the first stop (stop_sequence==0)
Other objective is to handle the case where the hours are >= 24 (25 for 1am, 26 for 2am…)

3rd step : Database creation

a/ create the train stops :
query on Neo4j

```
LOAD CSV WITH HEADERS FROM 'file:///stops.csv' AS line
CREATE (:Stop { stop_id: line.stop_id, stop_name: line.stop_name, stop_lat: line.stop_lat, stop_lon: line.stop_lon, parent_station: line.parent_station})
```

b/create an index on stop_id to quick things up

```
CREATE INDEX stop_name_index FOR (n:Stop) ON (n.stop_id)
```

c/create the pedestrian transfers between stations

We add monday=« 1 », tuesday =« 1 », since these transfers are available everyday

We also added 0.1 to each transfer, we will use this information to prevent our custom Dijkstra algorithm to propose two walks in a row (see implementation below).

query :

```
LOAD CSV WITH HEADERS FROM "file:///transfers.csv" AS csvLine
MATCH (start:Stop {stop_id: csvLine.from_stop_id }), (end:Stop {stop_id: csvLine.to_stop_id})

CREATE (start)-[:TransportationNew {end: toFloat(csvLine.new_arrival_time), saturday:"1", monday:"1", sunday:"1", tuesday:"1", wednesday:"1", thursday:"1", friday:"1",start_date:datetime(replace('20200101','ms','yyyyMMdd')) ,end_date: datetime(replace('20220101','ms','yyyyMMdd')) }]->(end)
```

d/ Create train transfers between stations

```
:auto USING PERIODIC COMMIT 10000
LOAD CSV WITH HEADERS FROM "file:///transits.csv" AS csvLine
```

```
MATCH (stop1:Stop {stop_id: csvLine.start_id}), (stop2:Stop {stop_id: csvLine.stop_id})
CREATE (stop1)-[:TransportationNew {stop_id: csvLine.stop_id,start_id:
csvLine.start_id,start_time_from_last_station: time(replace(csvLine.start_time_from_last_station,'ms','H
H:MM:SS')),route_id: csvLine.route_id,route_short_name: csvLine.route_short_name,route_type:
csvLine.route_type,monday: csvLine.monday,tuesday: csvLine.tuesday,wednesday: csvLine.wednesday,thursday:
csvLine.thursday,friday: csvLine.friday,saturday: csvLine.saturday,sunday: csvLine.sunday,start_date:
datetime(replace(csvLine.start_date,'ms','yyyyMMdd')),end _date:
datetime(replace(csvLine.end_date,'ms','yyyyMMdd')),durat ion: toFloat(csvLine.duration),
new_arrival_time:toFloat(csvLine.new_arrival_time),new_de parture_time:
toFloat(csvLine.new_departure_time),new_start_time_from_l
ast_station:toInteger(csvLine.new_start_time_from_last_st ation)}]->(stop2)
```

e/ Create a normed projection in order to have quicker results.

A normed projection is asking Neo4j to create a subset of the whole graph.

For example, we can create a projection for only train lines working on friday

In my example I will take Thursday between 18h (6pm) and above.

```
CALL gds.graph.create.cypher('transport18Hthursday2',
'MATCH (m:Stop) RETURN id(m) as id',
'MATCH (m1:Stop)-[r:TransportationNew]-(m2:Stop) where ((r. new_arrival_time > 151200 and

r.new_arrival_time < 159000) or(r.new_arrival_time < 4000.0)) and r.thursday="1" and r.start_date
<=datetime(replace("20210121","ms","yyyyMMdd")) and r.end_date >=
datetime(replace("20210121","ms","yyyyMMdd")) RETURN id(m1) as source, id(m2) as target, r.new_arrival_time
as new_arrival_time')
```

4th step: Modify original Dijkstra's algorithm from Neo4J in order for the algorithm to take into account only the timestamps at a given station that are created than the one before, in order to have reliable results

In order to do so :

clone https://github.com/neo4j/graph-data- science (BRANCH 1.4)

and replace the file ShortestPathDijkstra.java by mine provided in the repository

Compile the changed repository with ./gradlew packaging:shadowJar command and replace it in the GDS folder in Neo4j plugins.

5th step: Compare our results and CItyMapper's one :

To compare results at 6pm between stop « Porte de Saint cloud » and stop « Cluny la sorbonne », we use the cypher query :

```
MATCH (start:Stop {stop_id: 'StopPoint:59492'}), (end:Stop {stop_id: 'StopPoint:59:5067458'})

CALL gds.alpha.shortestPath.stream('transport18Hthursday', {

startNode: start,
endNode: end,
relationshipWeightProperty: 'new_arrival_time'

})
YIELD nodeId, cost
RETURN gds.util.asNode(nodeId).stop_name AS name, substring(toString((cost / 3600) - 24), 0, 2) + ':' +
toString(toInteger(round(toFloat(substring(toString((cost /3600)-24),2,4))*60))) as time
```

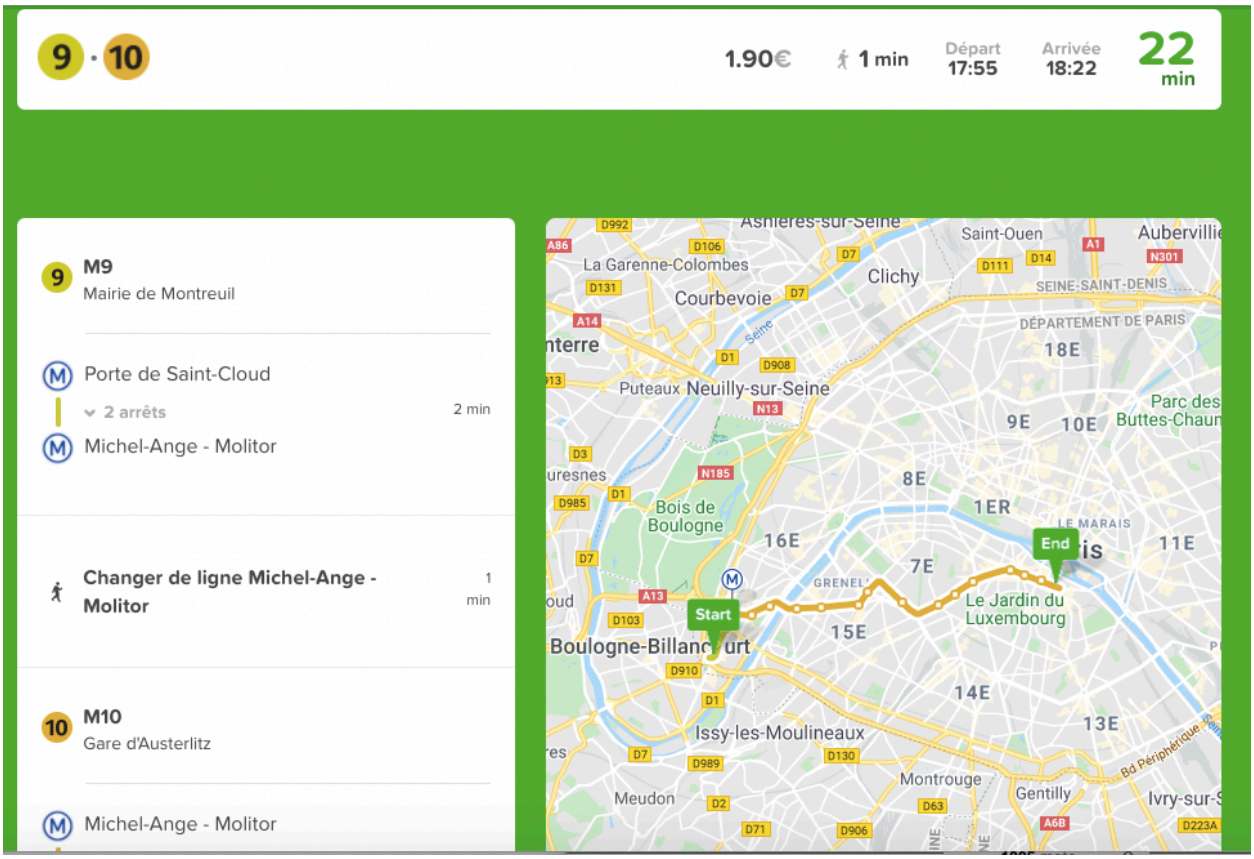
We have :

	name	time	
1	"Porte de Saint-Cloud"	"18:0"	
2	"Exelmans"	"18:1"	
3	"Michel-Ange-Molitor"	"18:2"	
4	"Michel-Ange-Molitor"	"18:8"	

5	"Chardon-Lagache"	"18:8"
6	"Mirabeau"	"18:9"
7	"Javel-André-Citroen"	"18:10"
8	"Charles Michels"	"18:11"
9	"Avenue Emile-Zola"	"18:13"
10	"La Motte-Picquet-Grenelle"	"18:15"
11	"Ségur"	"18:16"
12	"Duroc"	"18:18"
13	"Vaneau"	"18:19"
14	"Sèvres-Babylone"	"18:20"

15	"Mabillon"	"18:22"
16	"Odéon"	"18:23"
17	"Cluny-La Sorbonne"	"18:24"

So after leaving at 6pm (18h) we arrive at « cluny la sorbonne » at 6:24pm (18h24)



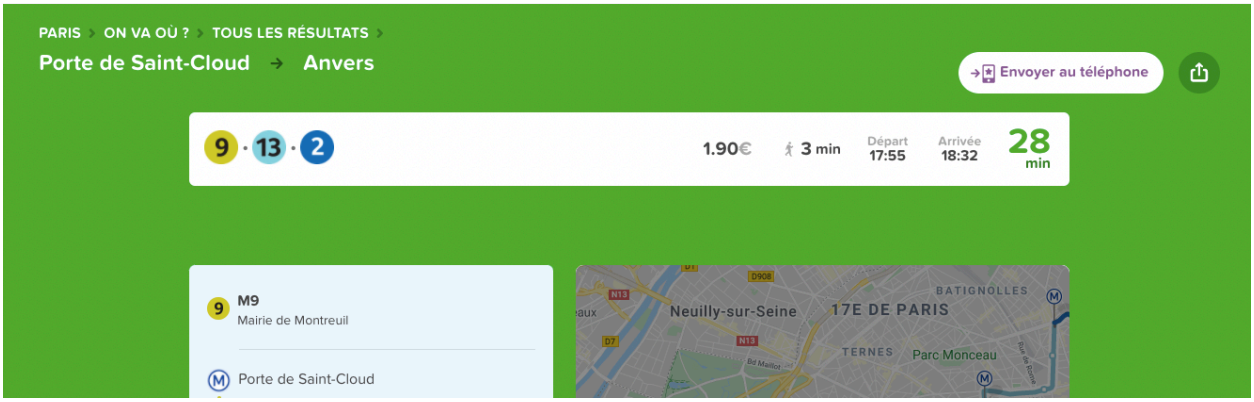
CityMapper provides the same itinerary and the time is almost the same (22mn)

Now with a more difficult itinerary : from « porte de saint Cloud » to « Anvers »

name	time	
1	"Porte de Saint-Cloud"	"18:0"
2	"Exelmans"	"18:1"
3	"Michel-Ange-Molitor"	"18:2"

4	"Michel-Ange-Auteuil"	"18:3"
5	"Jasmin"	"18:4"
6	"Ranelagh"	"18:6"

7	"La Muette"	"18:7"
8	"Rue de la Pompe (Avenue Georges Mandel)"	"18:8"
9	"Trocadéro"	"18:9"
10	"Iéna"	"18:10"
11	"Alma-Marceau"	"18:11"
12	"Franklin-Roosevelt"	"18:12"
13	"Saint-Philippe du Roule"	"18:14"
14	"Miromesnil"	"18:15"
15	"Miromesnil"	"18:19"
16	"Saint-Lazare"	"18:20"
17	"Liège"	"18:21"
18	"Place de Clichy"	"18:22"
19	"Place de Clichy"	"18:26"
20	"Blanche"	"18:28"
21	"Pigalle"	"18:29"
22	"Anvers"	"18:30"



Once again, same itinerary and only 2 min duration difference

The similarity between results are good, and the time to query Neo4j is good (124ms)

Next steps:
compare it with a postgresql (driver pgroutine)
doing the same with yenK algorithm to provide alternative results
create a web app : nix create-react-app better_than_citymapper, npm install, npm install neo4j- driver ....