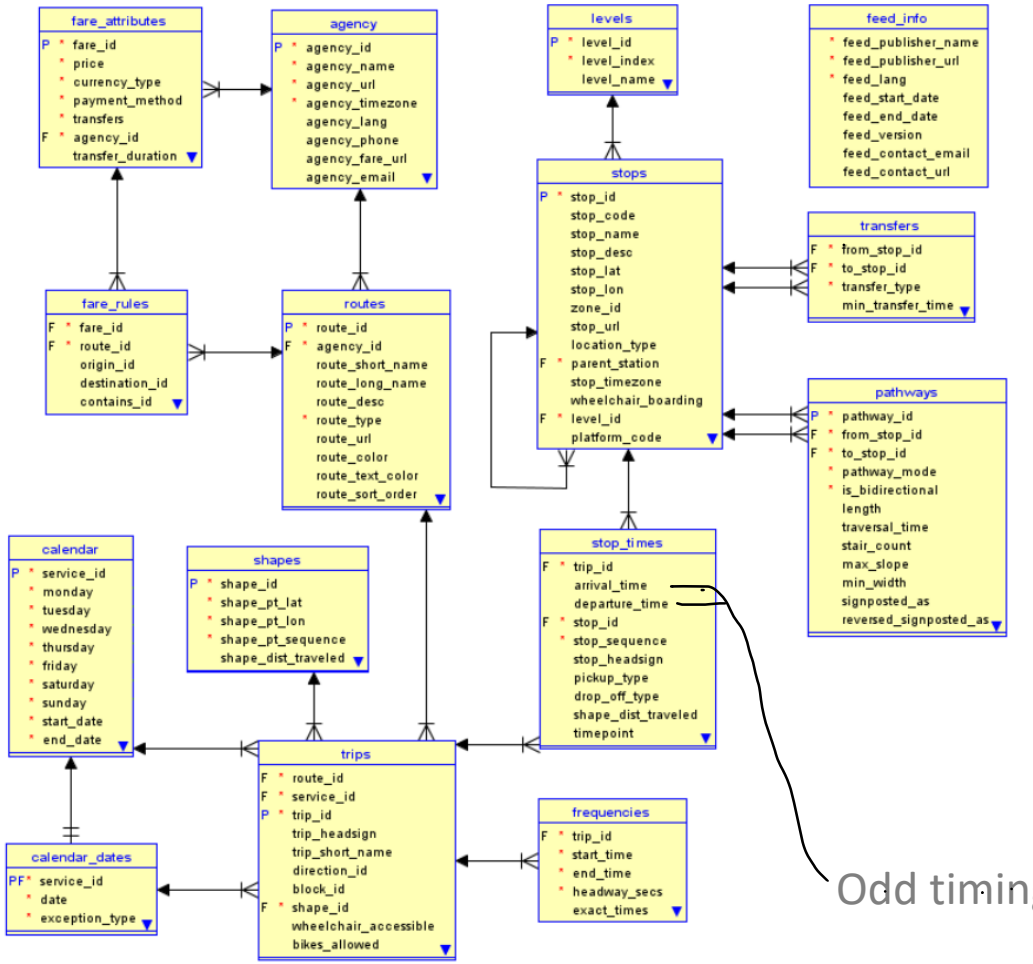
GTFS data modeling with Neo4j

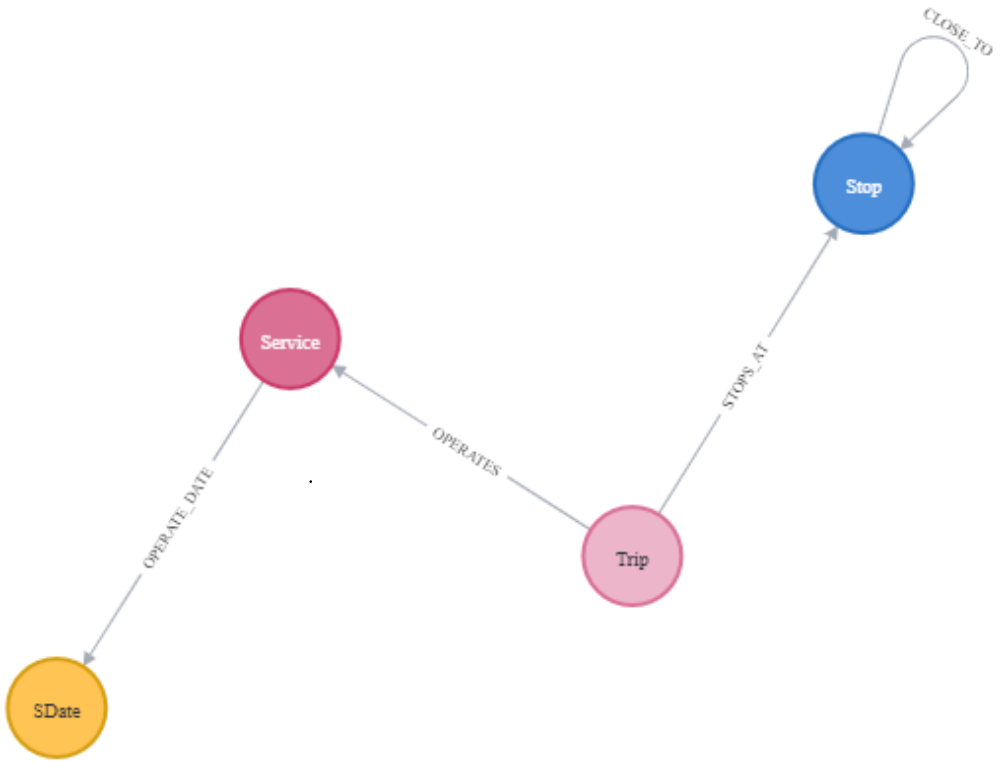
Bálint Hantos

Scientific databases and data modeling

Introduction



Odd timings: 24h+



TOOLS

Neo4j Desktop – for DBMS

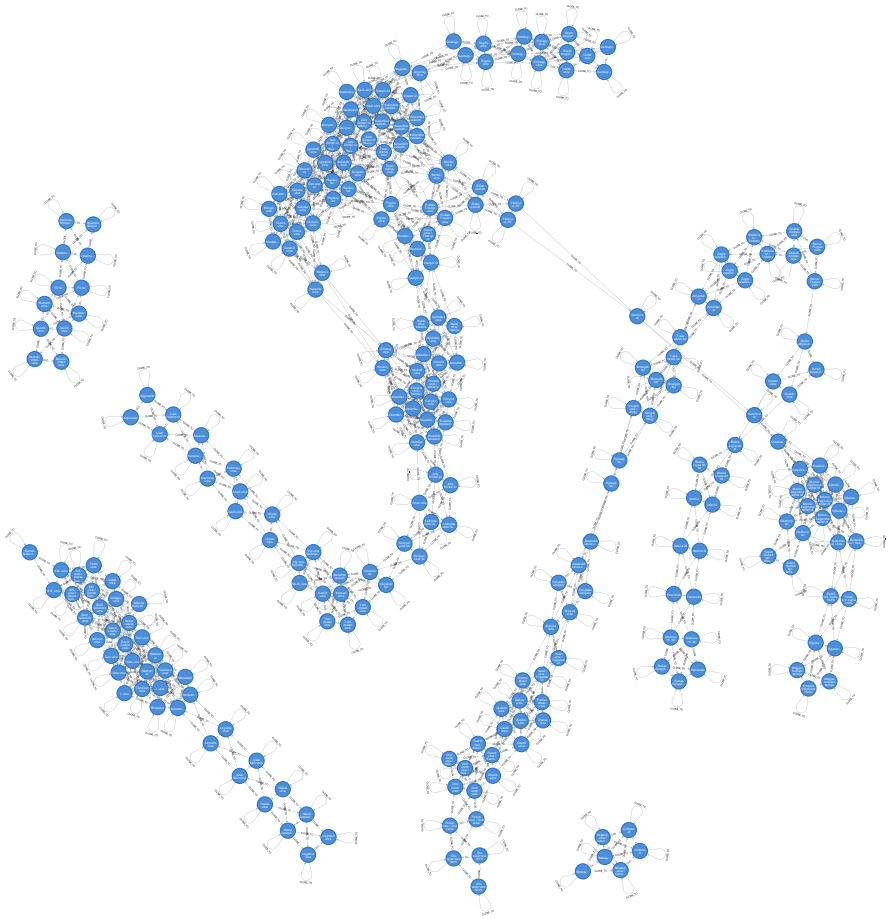
Cypher

+ APOC

(+ GDS)

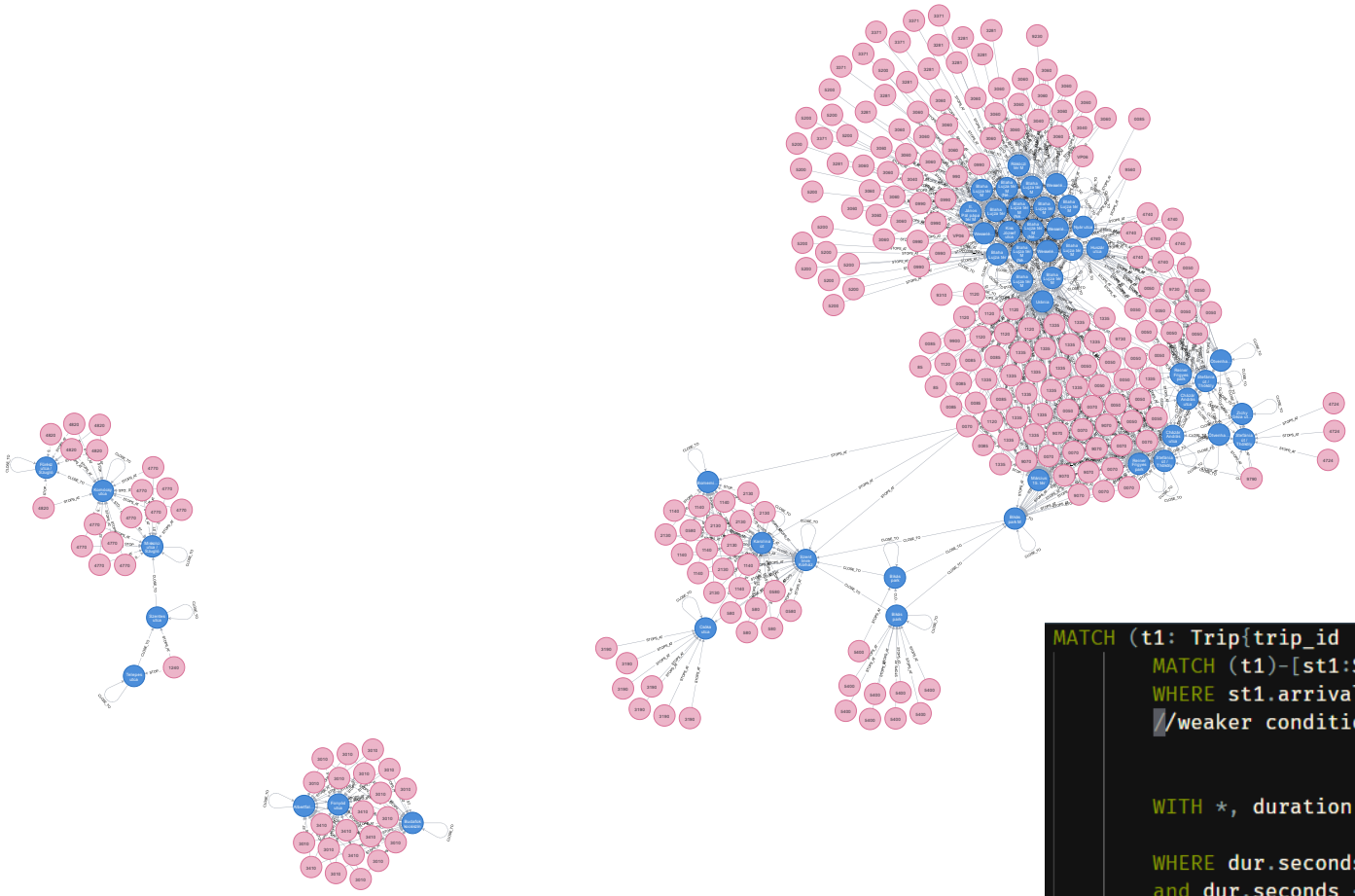
https://doi.org/10.33039/ami.2021.07.001

1: Reachable stops



```
MATCH (s1: Stop)
        WHERE s1.stop_name CONTAINS "Szent Imre Kórház"

        MATCH (s1)-[:STOPS_AT]-(t: Trip)-[:STOPS_AT]-(s2: Stop)
        RETURN DISTINCT s2
```

2: Switching rides



Precalculate distances

```
MATCH (t1: Trip{trip_id : "%s"})
        MATCH (t1)-[st1:STOPS_AT]-(: Stop)-[cls:CLOSE_TO]-(s2: Stop)-[st2:STOPS_AT]-(t2: Trip)
        WHERE st1.arrival_time < st2.departure_time
        //weaker condition, but it speeds up the calc a little


        WITH *, duration.inSeconds( st1.arrival_time, st2.departure_time ) as dur

        WHERE dur.seconds*1 > cls.d     // the stop must be in reachable distance
        and dur.seconds < 600          // make the switch in 10 minutes

        RETURN t2.trip_id, t2.route_id, s2.stop_name, dur.seconds
```

3: Path finding

- Minimum sum time: min(st2.arrive_time - st1.depart_time)

- Least amount of switches – shortestPath

```
MATCH (sd: SDate {service_date: date("2022-01-21")})-[*1]-(service_nodes)
WITH [ serv in COLLECT(service_nodes) | serv.service_id ] as services


MATCH path = (:Stop{stop_name: "Szent Imre Kórház"})-[:STOPS_AT]-(t1: Trip)-[:STOPS_AT|CLOSE_TO*0..3]-
(t2: Trip)- [:STOPS_AT]-(:Stop{stop_name: "Petőfi híd, budai hídfő"})

WHERE all( n in nodes(path) WHERE
            ( labels(n)<>["Trip"] or
              (n.service_id in services))
          )


WITH *, relationships(path) as rels
WHERE all( rel in rels WHERE
            ( type(rel)="CLOSE_TO" or
            ( time("08:00:00") <= rel.arrival_time <= time("09:00:00")) )
          )

WITH *, [ rel in rels | rel.arrival_time ] as arrtimes
WHERE apoc.coll.sort(arrtimes) = arrtimes  // check if trip goes in preferred direction
RETURN path
```

3: Path finding

- Minimal waiting time:
  - Travel time
  - Walking time
  - Waiting time

Sum of time

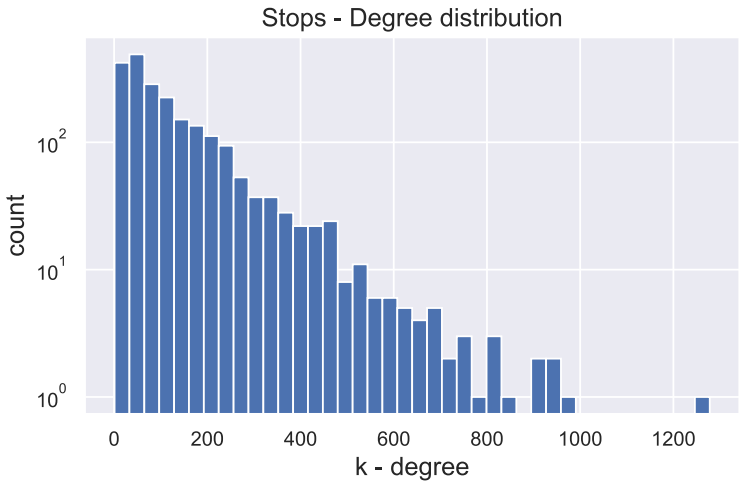
3: Path finding

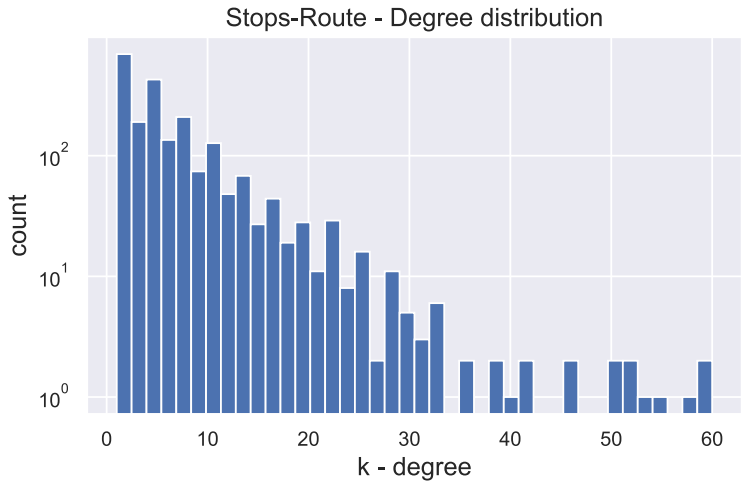
- Runs in finite time for 1-2 trip changes

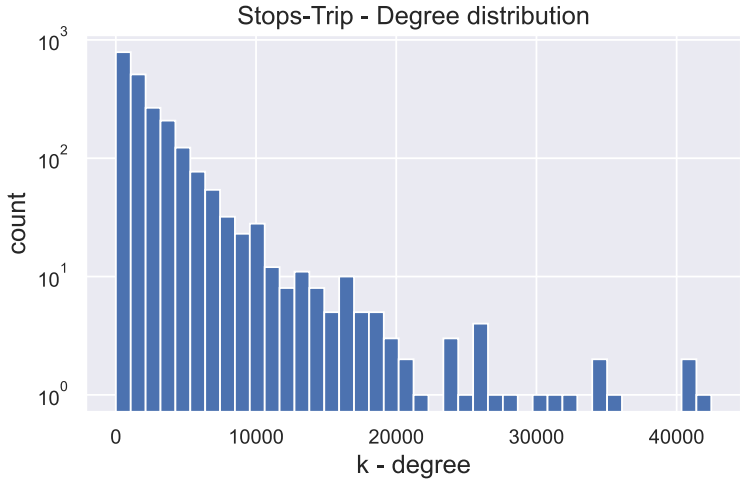
- Challenges:
  - Long runtimes
  - No results for >2 trips
  - Quasi pythonic syntax
  - Error messages refer to Java code

```
MATCH (sd: SDate {service_date: date("2022-01-21")})-[*1]-(service_nodes)
WITH [ serv in COLLECT(service_nodes) | serv.service_id ] as services


MATCH path = (:Stop{stop_name: "Szent Imre Kórház"})-[:STOPS_AT]-(t1: Trip)-[:STOPS_AT|CLOSE_TO*0..3]-
(t2: Trip)- [:STOPS_AT]-(:Stop{stop_name: "Petőfi híd, budai hídfő"})

WHERE all( n in nodes(path) WHERE
            ( labels(n)<>["Trip"] or
              (n.service_id in services))
          )


WITH *, relationships(path) as rels
WHERE all( rel in rels WHERE
            ( type(rel)="CLOSE_TO" or
            ( time("08:00:00") <=rel.arrival_time <= time("09:00:00")) )
          )

WITH *, [ rel in rels | rel.arrival_time ] as arrtimes
WHERE apoc.coll.sort(arrtimes) = arrtimes  // check if trip goes in preferred direction
RETURN path
```

5: Centrality measures







ERROR **Neo.ClientError.Procedure.ProcedureRegistrationFailed**

```
gds.graph.create.cypher is unavailable because it is sandboxed and has dependencies outside of the sandbox. Sandboxing is
controlled by the dbms.security.procedures.unrestricted setting. Only unrestrict procedures you can trust with access to
database internals.
```

Would be much needed for centrality calc and community detection