GTFS graph building and its path finding algorithms

isep

École d'ingénieurs du numérique

Summary

- Project overview.
- Our project architecture.
- Used technologies and external dependencies.
- How to load/build/persist the GTFS data graph ?
- How to search for a path in the GTFS graph ?
- How to cluster our GTFS graph ?
- Conclusion.

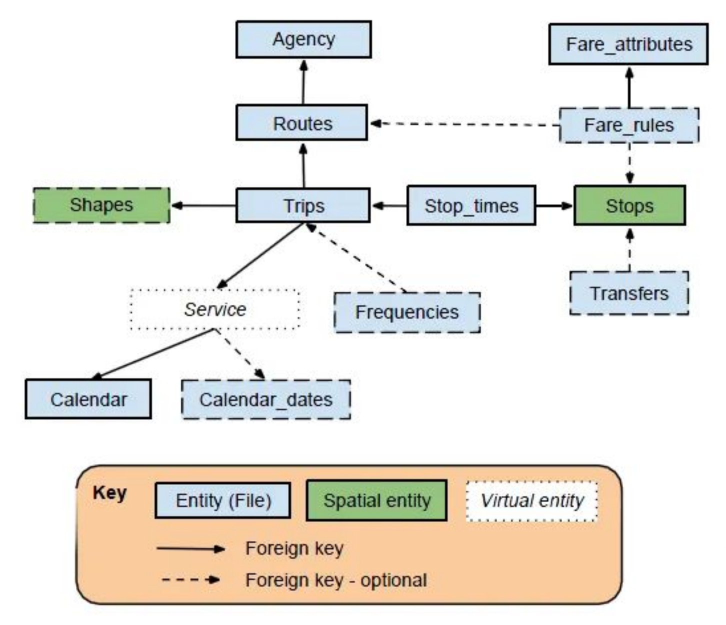
isep
École d'ingénieurs du numérique

Project overview - what is GTFS ?

- GTFS stands for "*General Transit Feed Specification*".

- A set of .csv (or .txt) files, containing data about a transportation network.

- Each file has potentially a foreign key field for it to be linked to another file. Just like the SQL tables.



École d'ingénieurs du numérique

Project overview - our goals

What we needed to do :

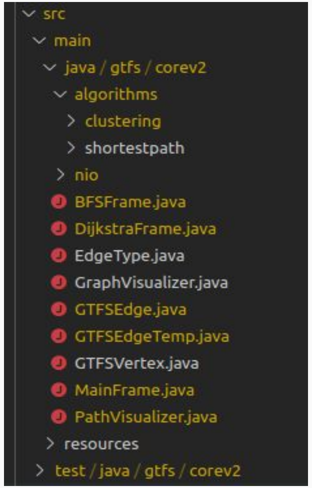
- Collection of data and construction of the graph
  - Unweighted graph
  - Weighted graph
- Calculation of shortest paths
- Shortest paths for graph clustering

Project architecture

- **algorithms :** with all the advanced algorithms algorithms.
  - **clustering :** find the clusters, their barycenters and the distance between one another.
  - **shortestpath** : contains the algorithms to find the shortest paths.

- **nio** : for non-blocking input output ; it contains the logic for loading, building and eventually serializing the GTFS data into a graph. It supports remote operations with a GCP bucket.

- **resources** : the local data are stored in the resources folder.
  - **mbta** : A folder to store the GTFS files of the mbta dataset.



```
v src
 v main
  v java / gtfs / corev2
   v algorithms
    > clustering
    > shortestpath
   > nio
   BFSFrame.java
   DijkstraFrame.java
   EdgeType.java
   GraphVisualizer.java
   GTFSEdge.java
   GTFSEdgeTemp.java
   GTFSVertex.java
   MainFrame.java
   PathVisualizer.java
  > resources
 > test / java / gtfs / corev2
```

isep
École d'ingénieurs du numérique

Used technologies and external dependencies

Google Cloud Platform

Java™

mxGraph

Gradle

JGraphT

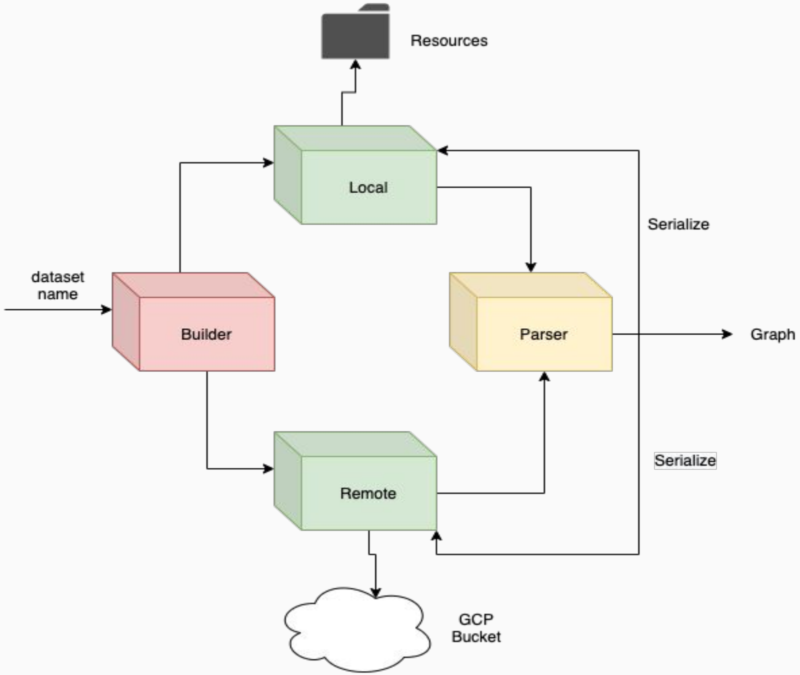
JUnit

eclipse

isep

École d'ingénieurs du numérique

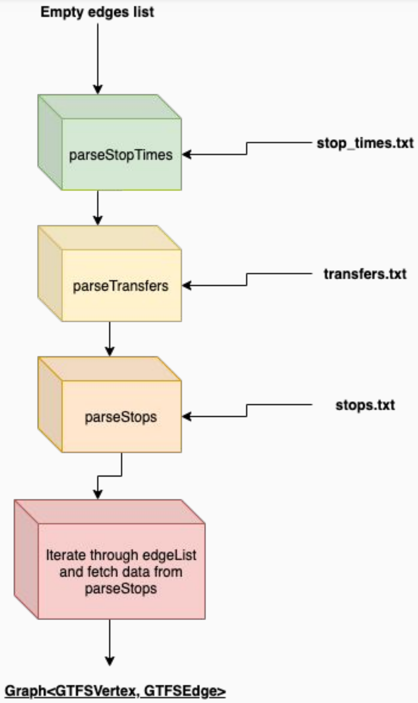
How to load/build/persist the GTFS data graph ?



- **Builder** design pattern (*GTFSGraphBuilder class*)
- Can fetch data from local/remote destination in a multithreaded way.
- Parse the raw data loaded in memory and generate the *GTFSGraph<GTFSVertex, GTFSEdge>* object.
- Ability to serialize/unserialize (keeping the data integrity) the graph object under JSON format for a faster retrieval the next time (**100x** speed in local, **1000x** speed in remote)

isep
École d'ingénieurs du numérique

How to parse GTFS data ?



- Generate a List<List<String>> representing the edge list in parsing **transfers.txt** and **stop_times.txt**
  - use **parseStopTimes(edgeList)** function to mutate a Map<String, GTFSEdgeTemp> where the key is a code containing the starting stop id and the target stop id.
  - use **parseTransfers(edgeList)** function to mutate a Map<String, GTFSEdgeTemp> where the key is a code containing the starting stop id and the target stop id. The transfers add the travel within the same station whereas **parseStopTimes** is for two different stations.
- Generate a Map<String, List<String>> which is a representation of **stops.txt** where the key is the stop id and the value is the remaining data in the line
- Iterate through **edgeList** and for each edges find the corresponding vertex data (latitude, longitude and names in our simplified version) using the Map<String, List<String>> computed earlier. Create the real GTFS graph with weighted edges computed with the distance in km between two vertex.

isep
École d'ingénieurs du numérique

How to search for a path in the GTFS graph ?

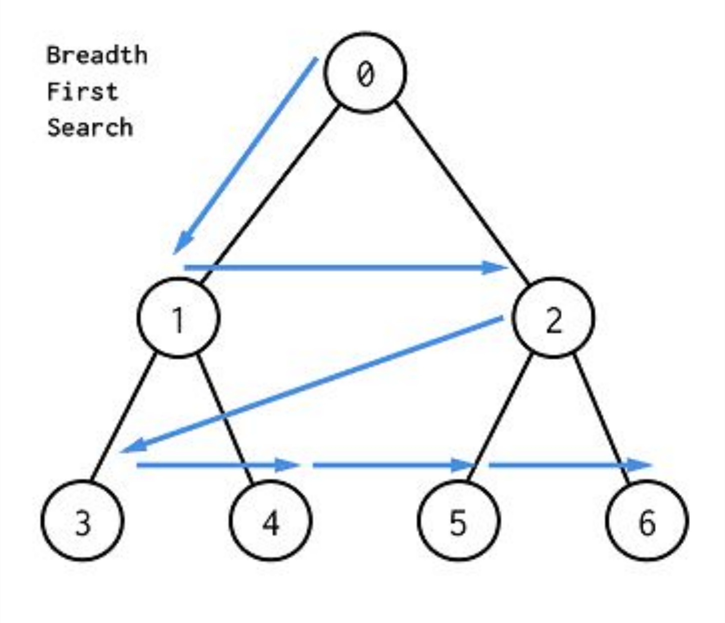
- Breadth First Search (BFS) in order to search for the best path in an Unweighted Graph
- Dijkstra to find shortest path in terms of distance in Weighted Graph
- Find path between two nodes or clusters

BFS

Nodes that have already been visited are marked to prevent the same node from being explored more than once.

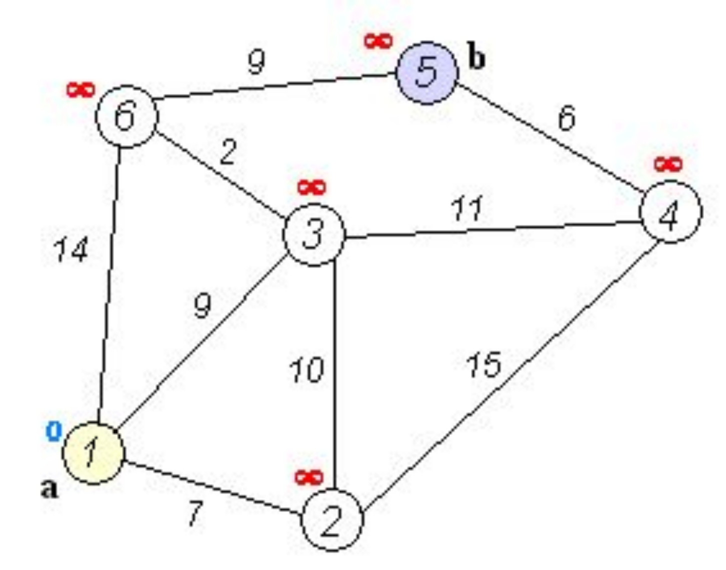
Algorithm Steps :

- Put the source node in the queue.
- Remove the node from the beginning of the queue for processing.
- Put all unexplored neighbors in the queue (at the end).
- If the queue is not empty, resume step 2.

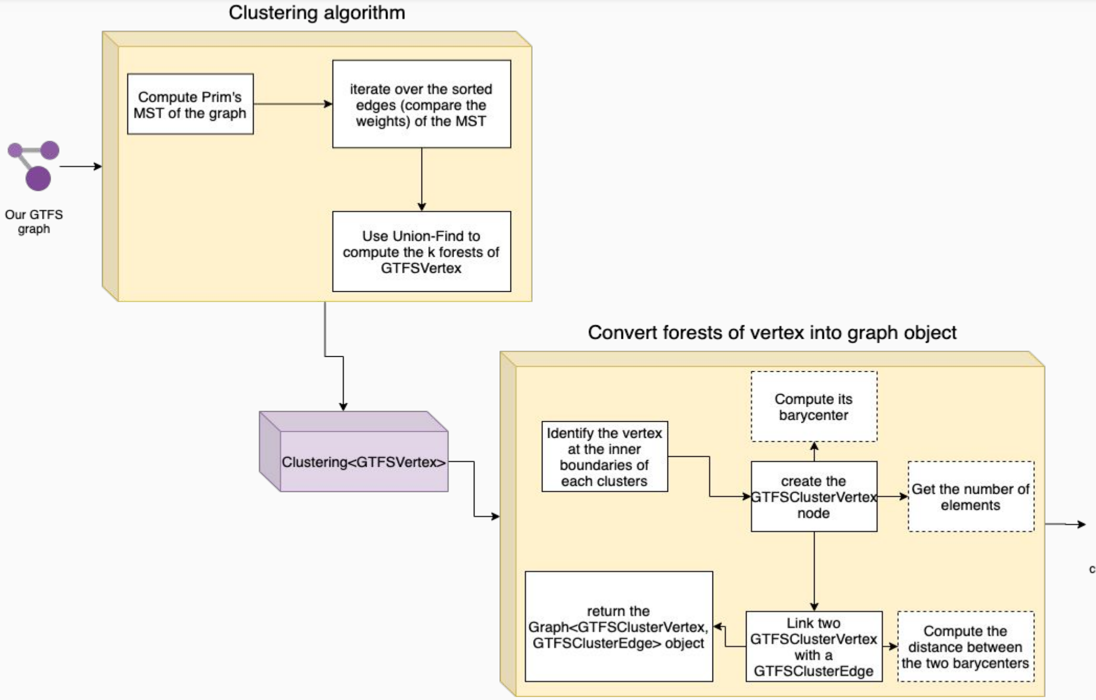


Dijkstra

- Build a sub-graph in which the different vertices are classified in ascending order of their minimum distance from the starting vertex.
- At the beginning, we consider that the distances from each vertex to the starting vertex are infinite
- During each iteration, a vertex of minimum distance is chosen outside the subgraph and added to the subgraph.
- The distances of the vertices adjacent to the added one are updated



How to cluster our GTFS graph ?



Clustering algorithm

Our GTFS graph

Compute Prim's MST of the graph

iterate over the sorted edges (compare the weights) of the MST

Use Union-Find to compute the k forests of GTFSVertex

Clustering<GTFSVertex>

Convert forests of vertex into graph object

Identify the vertex at the inner boundaries of each clusters

Compute its barycenter

create the GTFSClusterVertex node

Get the number of elements

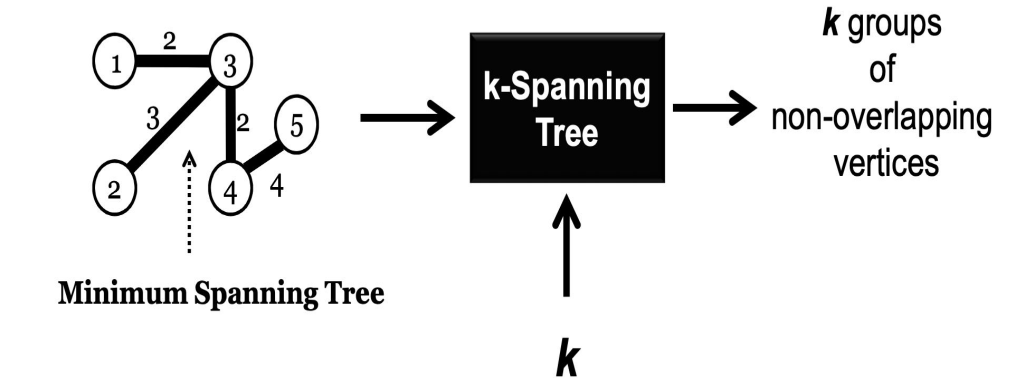
Link two GTFSClusterVertex with a GTFSClusterEdge

Compute the distance between the two barycenters

return the Graph<GTFSClusterVertex, GTFSClusterEdge> object

Our GTFS cluster graph

How to cluster our GTFS graph ?

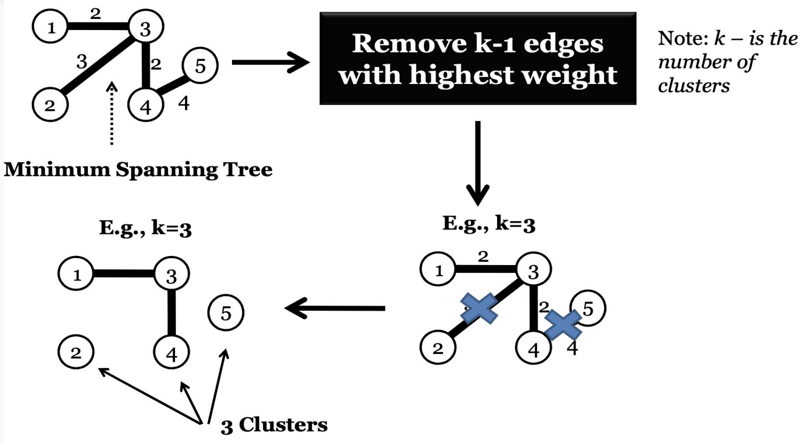


Minimum Spanning Tree

k-Spanning Tree

k groups of non-overlapping vertices

k



Minimum Spanning Tree

Remove k-1 edges with highest weight

Note: *k – is the number of clusters*

E.g., k=3

E.g., k=3

3 Clusters

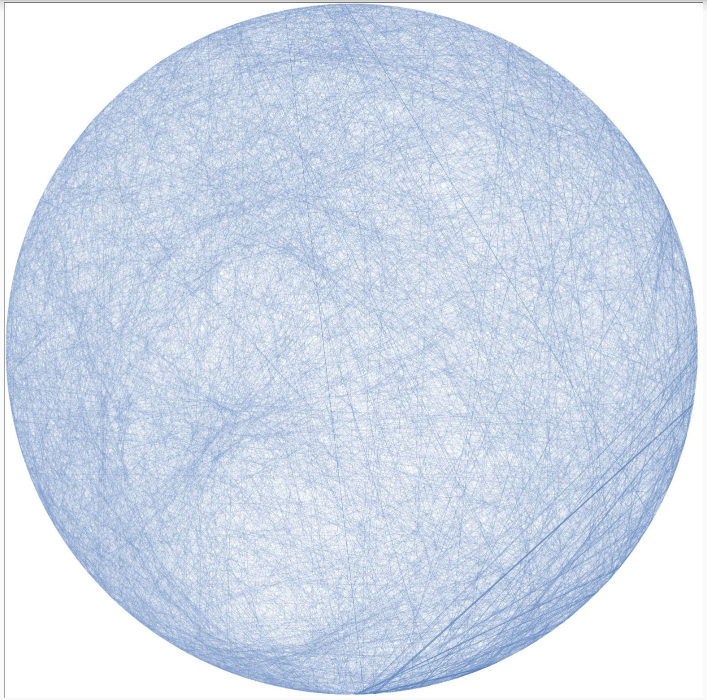
- Obtains the MST of the input graph.
- Obtains the list of sorted edges of the MST
- Remove k -1 edges from the MST. (use of Union-Find)
- Results in k cluster. (The k partition of the Union-Find)

Complexity of our cluster algorithm

$$O(\underbrace{m + nlog(n)}_{\text{Prim’s MST}} + \underbrace{nlog(n)}_{\text{Sort MST’s edges}} + \underbrace{klog^*(n)}_{\text{Build the forests}} + (\underbrace{kn}_{\text{Compute barycenters for each cluster}} + \underbrace{(n-k)m}_{\text{Link the disjoint clusters}}))$$

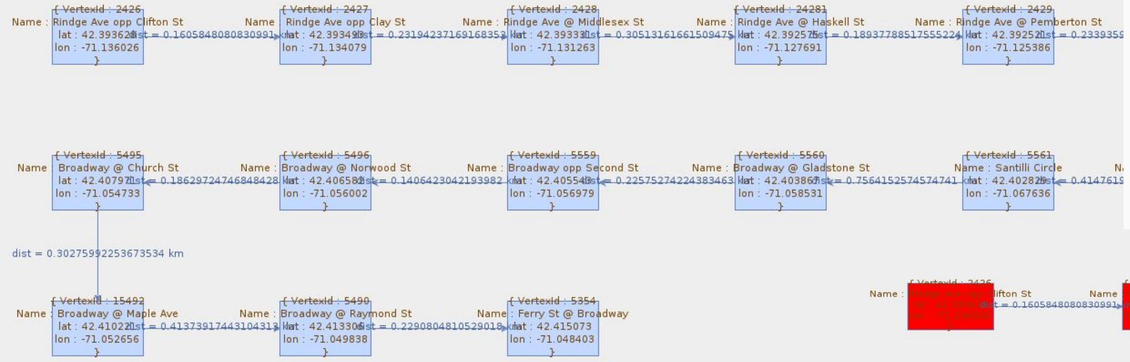
$$O((n-k)m)$$

Conclusion



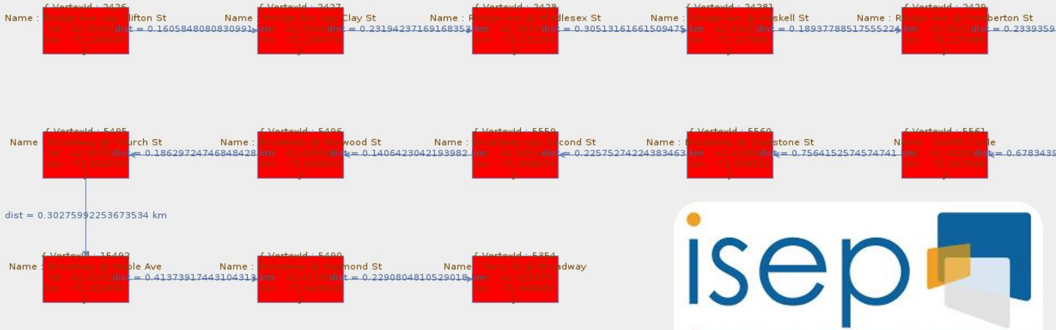
- Drawings of boston public transit network
- Almost 8000 vertices 11000 edges.

Conclusion



```
{ VertexId : 2426
Name : Rindge Ave opp Clifton St
lat : 42.393628
lon : -71.136026
}
dist = 0.1605848080830991 km

{ VertexId : 2427
Name : Rindge Ave opp Clay St
lat : 42.393493
lon : -71.134079
}
dist = 0.23194237169168353 km

{ VertexId : 2428
Name : Rindge Ave @ Middlesex St
lat : 42.393331
lon : -71.131263
}
dist = 0.30513161661509475 km

{ VertexId : 24281
Name : Rindge Ave @ Haskell St
lat : 42.392575
lon : -71.127691
}
dist = 0.18937788517555224 km

{ VertexId : 2429
Name : Rindge Ave @ Pemberton St
lat : 42.392521
lon : -71.125386
}
dist = 0.2339359

{ VertexId : 5495
Name : Broadway @ Church St
lat : 42.407971
lon : -71.054733
}
dist = 0.18629724746848428 km

{ VertexId : 5496
Name : Broadway @ Norwood St
lat : 42.406582
lon : -71.056002
}
dist = 0.1406423042193982 km

{ VertexId : 5559
Name : Broadway opp Second St
lat : 42.405548
lon : -71.056979
}
dist = 0.22575274224383463 km

{ VertexId : 5560
Name : Broadway @ Gladstone St
lat : 42.403867
lon : -71.058531
}
dist = 0.7564152574574741 km

{ VertexId : 5561
Name : Santilli Circle
lat : 42.402829
lon : -71.067636
}
dist = 0.4147619

dist = 0.30275992253673534 km

{ VertexId : 15492
Name : Broadway @ Maple Ave
lat : 42.410221
lon : -71.052656
}
dist = 0.41373917443104313 km

{ VertexId : 5490
Name : Broadway @ Raymond St
lat : 42.413306
lon : -71.049838
}
dist = 0.2290804810529018 km

{ VertexId : 5354
Name : Ferry St @ Broadway
lat : 42.415073
lon : -71.048403
}
```

- Drawings of the path between two nodes using BFS and Dijkstra


Name : lifton St
dist = 0.1605848080830991 km
Name : Clay St
dist = 0.23194237169168353 km
Name : R dlesex St
dist = 0.30513161661509475 km
Name : skell St
dist = 0.18937788517555224 km
Name : R berton St
dist = 0.2339359
Name : urch St
dist = 0.18629724746848428 km
Name : wood St
dist = 0.1406423042193982 km
Name : cond St
dist = 0.22575274224383463 km
Name : stone St
dist = 0.7564152574574741 km
N le
dist = 0.6783439
dist = 0.30275992253673534 km
Name : ple Ave
dist = 0.41373917443104313 km
Name : mond St
dist = 0.2290804810529018 km
Name adway