Automatic Generation of Frequency Maps for Public Transit Networks

Supervisors

Prof. Dr. Hannah Bast

Prof. Dr. Georg Lausen

Patrick Brosi

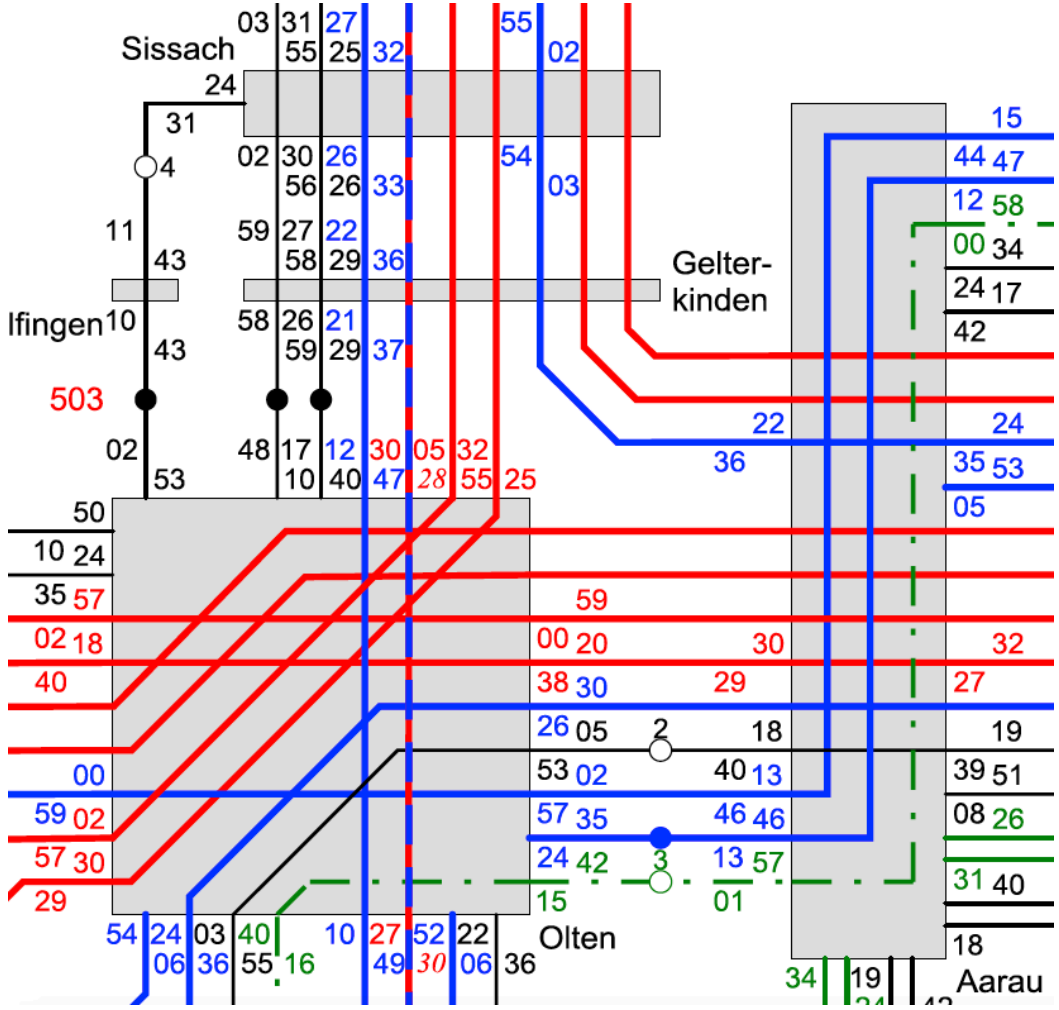
Bhashitha Gamage

Department of Computer Science

Albert-Ludwigs-University Freiburg

30th of May, 2018

Motivation



Section of the manually drawn Official Swiss Railway frequency map [1]

Goal

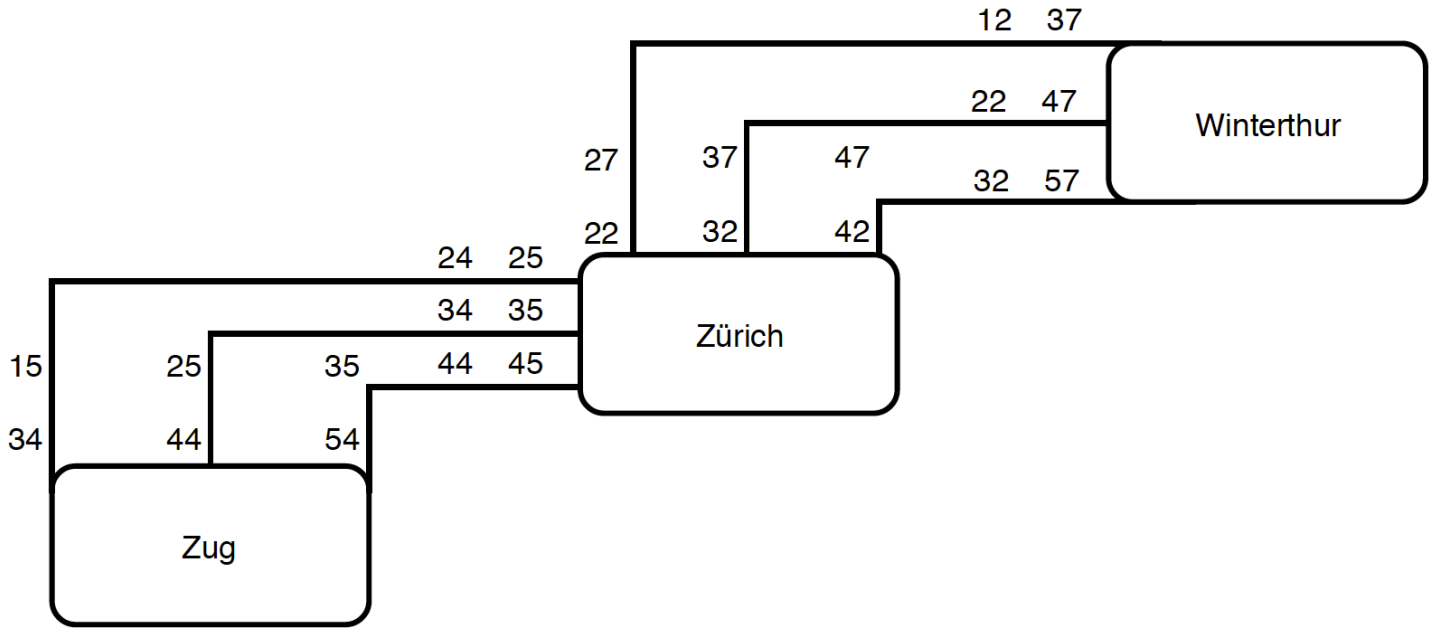
- Automatic generation of frequency maps that can be used as blueprints for the manual drawing of frequency maps.

Outline

- Frequency Maps
- GTFS data
- Drawing of Frequency Maps
    1. Extraction of the transit graph from GTFS data
    2. Implementation of the Frequency Finding Algorithm
    3. Extraction of the frequency coverages from interesting nodes
    4. Drawing of the frequency graph in convenient manner
- Evaluation
- Summary of the contribution
- Future Work

Frequency Maps

- Frequency Maps give information about the frequency coverage of transport medium in a public transit network



Sample frequency graph

- Ex: Every hour 15th minute a train is leaving from Zug and it arrives to Zürich at 25th minute
  Every hour 24th minute a train is leaving from Zürich it arrives to Zug at 34th minute

GTFS Data

A General Transit Feed Specification is a collection of series of comma separated text files.

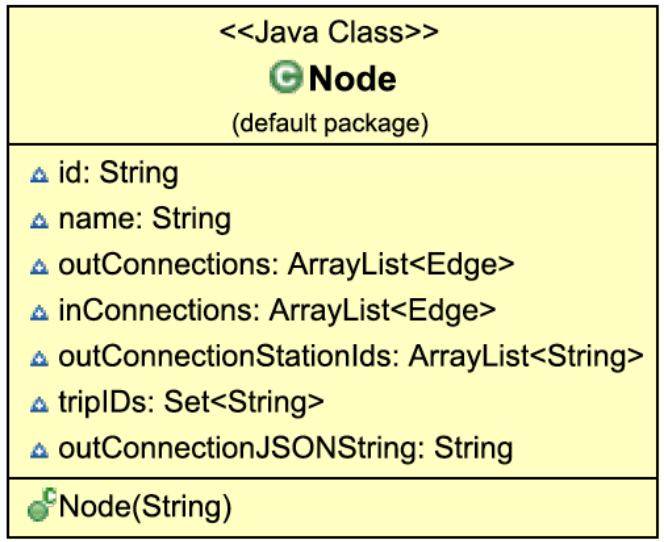
Filename	Required	Defines
agency.txt	**Required**	One or more transit agencies that provide the data in this feed.
stops.txt	**Required**	Individual locations where vehicles pick up or drop off passengers.
routes.txt	**Required**	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	**Required**	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	**Required**	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	**Required**	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
calendar_dates.txt	Optional	Exceptions for the service IDs defined in the calendar.txt file. If calendar.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
fare_attributes.txt	Optional	Fare information for a transit organization's routes.
fare_rules.txt	Optional	Rules for applying fare information for a transit organization's routes.
shapes.txt	Optional	Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt	Optional	Headway (time between trips) for routes with variable frequency of service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
feed_info.txt	Optional	Additional information about the feed itself, including publisher, version, and expiration information.

Google GTFS reference page [2]

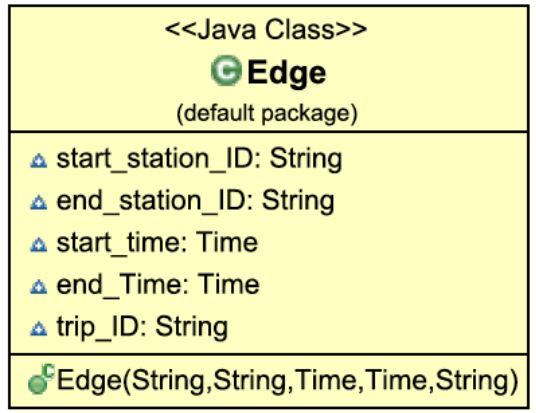
1. Extraction of the transit graph from the GTFS Data

```
trip_id,arrival_time,departure_time,stop_id,stop_sequence,stop_headsign,
pickup_type,drop_off_type,shape_dist_traveled,attributes_ch
1,11:42:00,11:42:00,8050807,0,,0,0,,
1,11:46:00,11:46:00,8050806,1,,0,0,,
1,11:52:00,11:52:00,8050805,2,,0,0,,
1,11:56:00,11:56:00,8050804,3,,0,0,,
1,11:59:00,11:59:00,8050803,4,,0,0,,
1,12:05:00,12:05:00,8050802,5,,0,0,,
1,12:12:00,12:12:00,8091916,6,,0,0,,
```

stop_times.txt [3]



Node class



Edge class

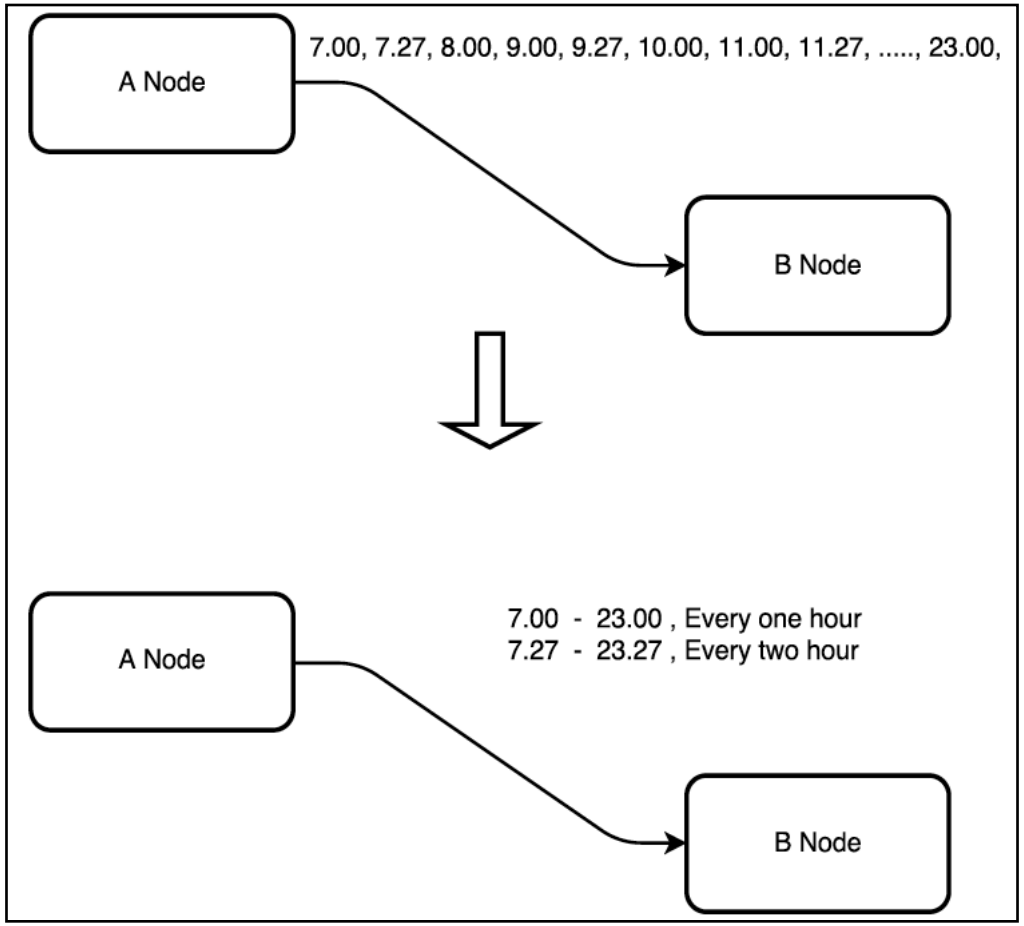
1. Extraction of the transit graph from the GTFS Data cont.

- Transit graph

```
Node: 8000339
In connections
 Start Station 8002238 arrival Time 00:04:00 Trip ID: 8582:1, Service ID: 8582:1:1:s
 Start Station 8002238 arrival Time 00:04:00 Trip ID: 8582:2, Service ID: 8582:2:2:s
 Start Station 8002238 arrival Time 00:04:00 Trip ID: 8582:3, Service ID: 8582:3:3:s
 Start Station 8000988 arrival Time 04:52:00 Trip ID: 8583, Service ID: 8583:1:s
 Start Station 8000988 arrival Time 04:53:00 Trip ID: 8584, Service ID: 8584:1:s
 Start Station 8002238 arrival Time 05:04:00 Trip ID: 8585, Service ID: 8585:1:s

Out connections
 End Station 8000988 depart Time 00:04:00 Trip ID: 8582:1, Service ID: 8582:1:1:s
 End Station 8000988 depart Time 00:04:00 Trip ID: 8582:2, Service ID: 8582:2:2:s
 End Station 8000988 depart Time 00:04:00 Trip ID: 8582:3, Service ID: 8582:3:3:s
 End Station 8002238 depart Time 04:52:00 Trip ID: 8583, Service ID: 8583:1:s
 End Station 8002238 depart Time 04:53:00 Trip ID: 8584, Service ID: 8584:1:s
 End Station 8000988 depart Time 05:04:00 Trip ID: 8585, Service ID: 8585:1:s
 End Station 8002238 depart Time 05:53:00 Trip ID: 8586:1, Service ID: 8586:1:1:s
```

Transit graph with out-connections and in-connections

2. Implementation of the Frequency Finding Algorithm



Two nodes with set of departure times

2. Implementation of the Frequency Finding Algorithm cont.

- Main Goal:

  Finding of the arithmetic progressions given a set of departure times

- Arithmetic progressions can be represented as frequency labels

- The algorithm is adapted from the ”Frequency-Based Search for Public Transit” research paper by Prof Dr Hannah Bast and Sabine Storandt

2. Implementation of the Frequency Finding Algorithm cont.

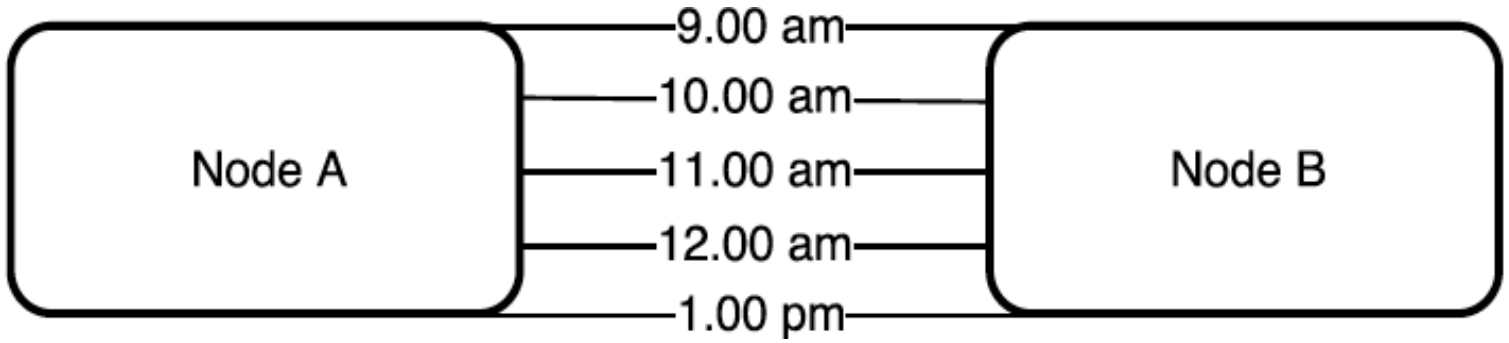
- Frequency finding algorithm
  - Starts with the smallest departure t1 and search for the longest arithmetic progression (AP) starting with t1
  - Add the AP to a collection and mark all elements covered by the AP
  - Then repeat the approach with the next unmarked element t2 as start time
- Running time of the algorithm is $O(N^3)$

2. Implementation of the Frequency Finding Algorithm cont.

- Improved version of the frequency finding algorithm
  - Introduce minimum AP length (K) which reduce iteratively
- Modifications
  - Human friendly frequency finding
  - Introduction of boundary filtering

3. Extraction of the frequency coverage between interesting nodes

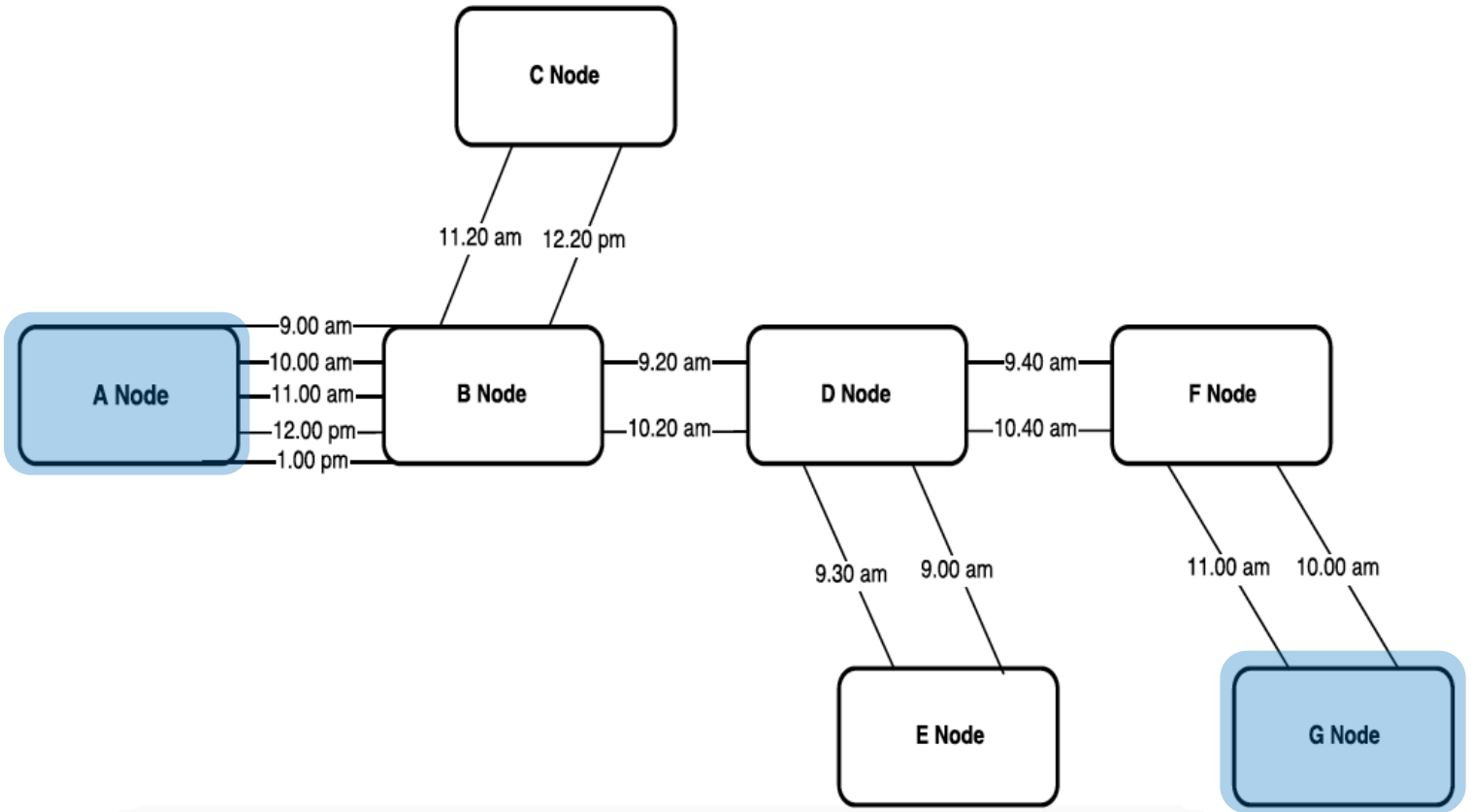
- Finding the frequency coverage between the two consecutive nodes is trivial



The diagram with two consecutive nodes named A and B and out connections from Node A to Node B

3. Extraction of the frequency coverage between interesting nodes cont.

- How to find the frequency coverage of two distantly located nodes A and G?



The diagram with multiple nodes and out connections

3. Extraction of the frequency coverage between interesting nodes cont.

- Approach one:
  - Navigate through all the out connections of node A and go to the next nodes
  - Then navigate through all the out connections of that node again
  - …
  - Until reach the node G

- Running time depends on the # of out connections and # of intermediate nodes

3. Extraction of the frequency coverage between interesting nodes cont.

- Approach two:
  - Retrieves the tripIDs of the trips which covers each of these node
  - Get the intersection of the tripIDs
  - Checks for the direction of the trip and collects the tripIDs into a collection
  - Retrieves the departure times of each trip from Node A, the travel duration and store them in collections
  - Sort the staring times collection in ascending order, and feed to frequency finding algorithm

4. Drawing of the frequency lines and nodes which resembles a schematic map

- Experimented with four approaches
  - Web application using Leaflet
  - QGIS using GeoJSON
  - Octi Tool
  - Graphviz

01) Web Application using Leaflet

- Uses the client-server architecture
  - Server is implemented in Java
  - Client is a web page embedded with Leaflet map view
  - Communication takes place using get requests and JSON objects
  - Demo

02) QGIS Using GeoJSON

- QGIS: Quantum Geographic Information System
- GeoJson is a JSON format which is used to describe geographical features
  - Nodes: Points
  - Frequency Lines: Line strings


```
{
   "type": "FeatureCollection",
   "features": [{
      "type": "Feature",
      "geometry": {
      "type": "Point",
      "coordinates": [47.586826, 7.636695]
   },
      "properties": {
      "name": "Weil am Rhein",
      "id": "8014428"
   }
   }]
}
```


Ex for GeoJson node explanation

02) QGIS Using GeoJSON cont.

The Switzerland railway map rendered by QGIS with web mercator coordinates

03) Octi Tool

- Octi is a tool developed under the chair for Algorithms and Data Structures
- Can renders maps, using GeoJSON data
- Snaps station nodes to nodes on an octi-linear grid graph
- Every node is connected by 45, 135, 225 and 315 degrees edges to its direct neighbors

04) Graphviz

- An open source graph drawing tool
- Can draw graphs specifies in dot language scripts

Dot Language

- Dot is a graph description language
- Dot graphs are files with gv or dot extension
- Programs that can process Dot files
  - dot
  - neato
  - fdp etc.

04) Graphviz cont.

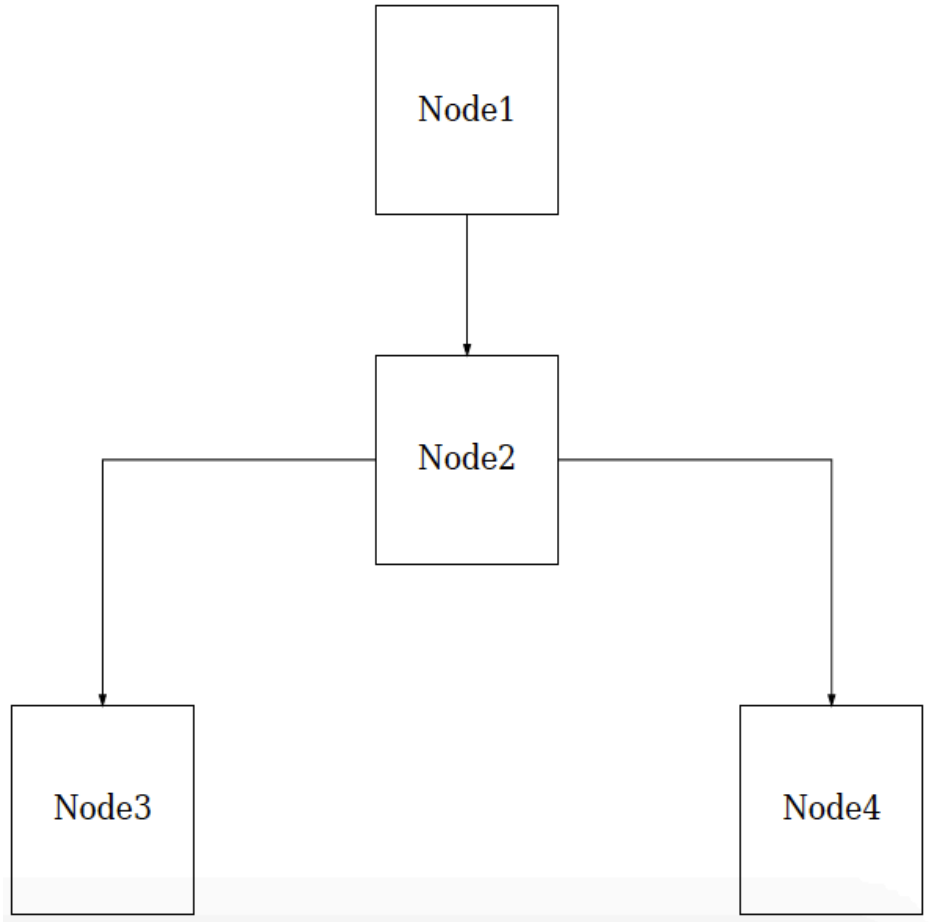
Neato layout engine

- "spring model" layouts and attempts to minimize a global energy function - default behavior
- Can position the nodes
- Orthogonal edge style is supported

04) Graphviz cont.

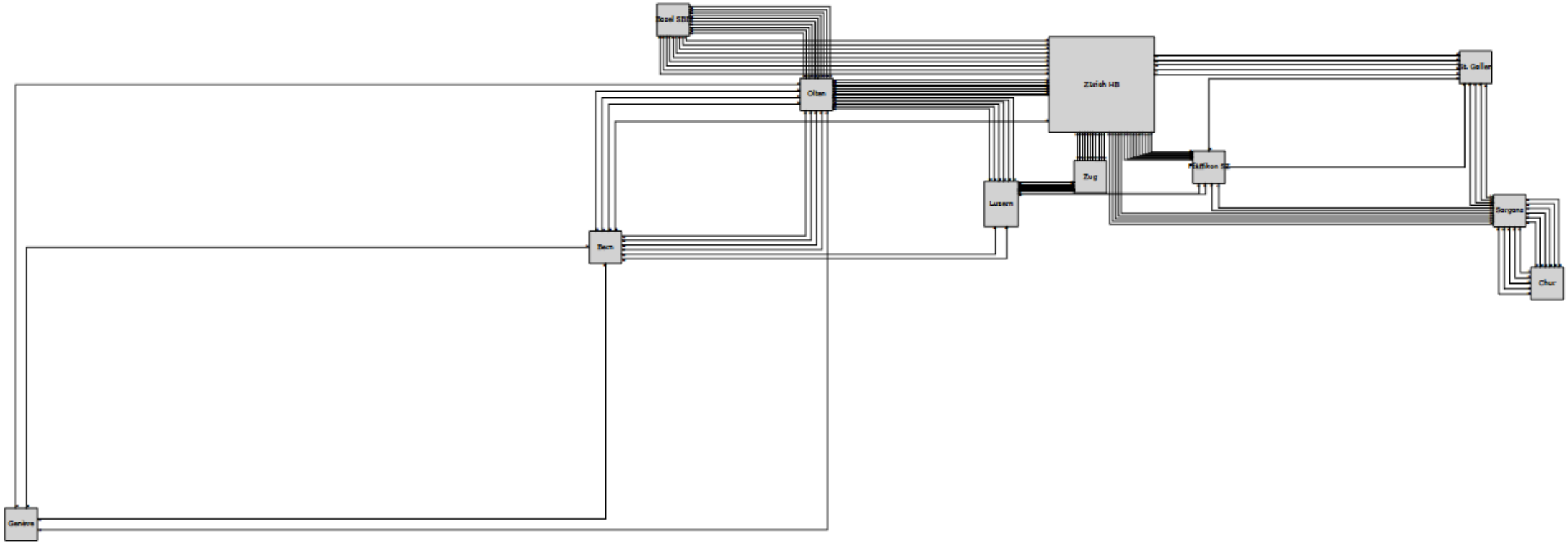
```
digraph g {
   splines=ortho;
   Node1 [pos = "10,15!",fontsize=35, shape = box,width=3, height=3 ,label="
      Node1" ];
   Node2 [pos = "10,10!",fontsize=35, shape = box,width=3, height=3 ,label="
      Node2" ];
   Node3 [pos = "4,5!",fontsize=35, shape = box,width=3, height=3, label="
      Node3" ];
   Node4 [pos = "16,5!",fontsize=35, shape = box,width=3, height=3 ,label="
      Node4" ];
   Node1 -> Node2 -> Node3;
   Node2 -> Node4;
}
```

Graph described in Dot language



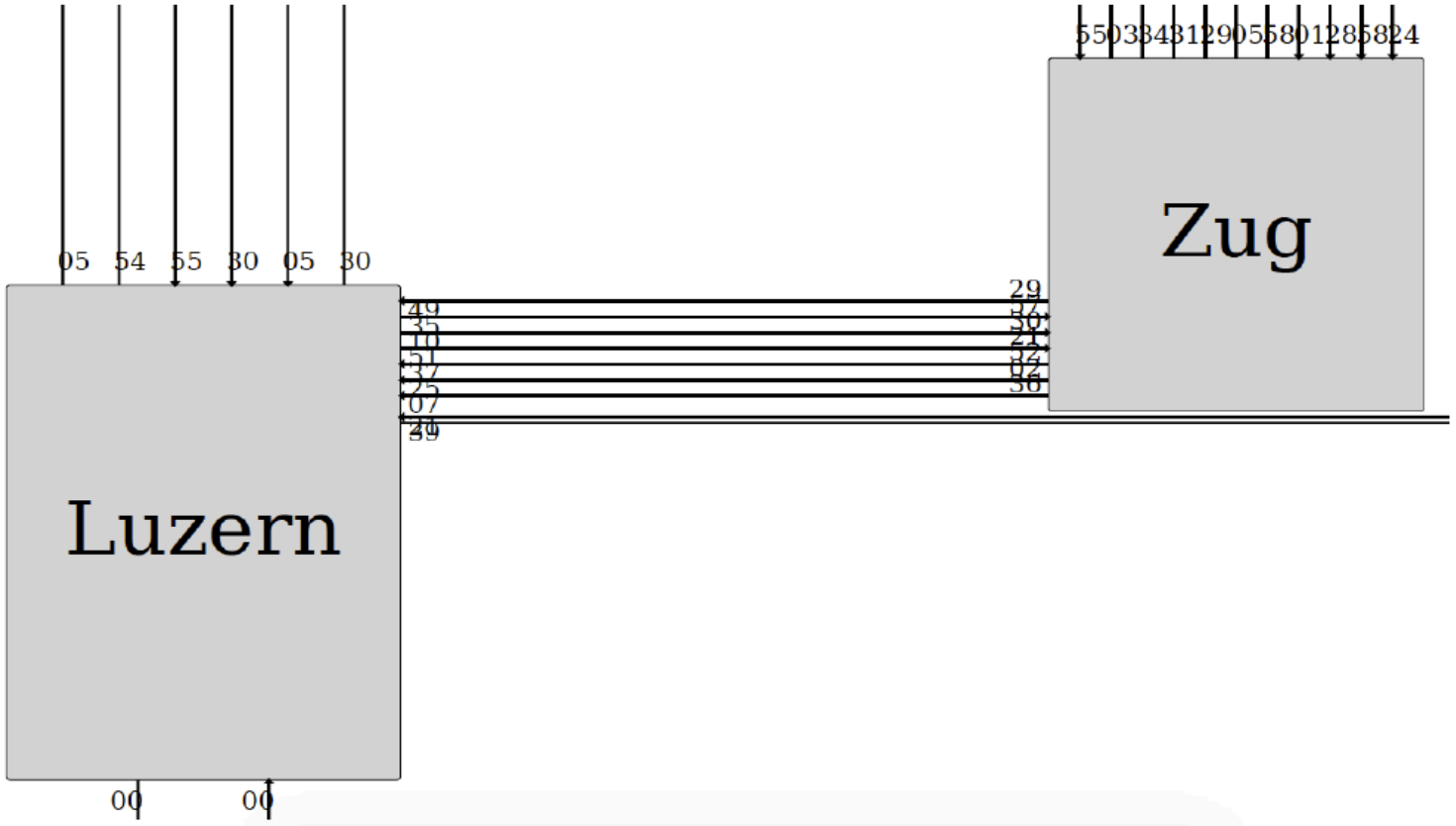
Graph rendered by neato layout engine

04) Graphviz cont.



Initial frequency graph rendered by Graphviz

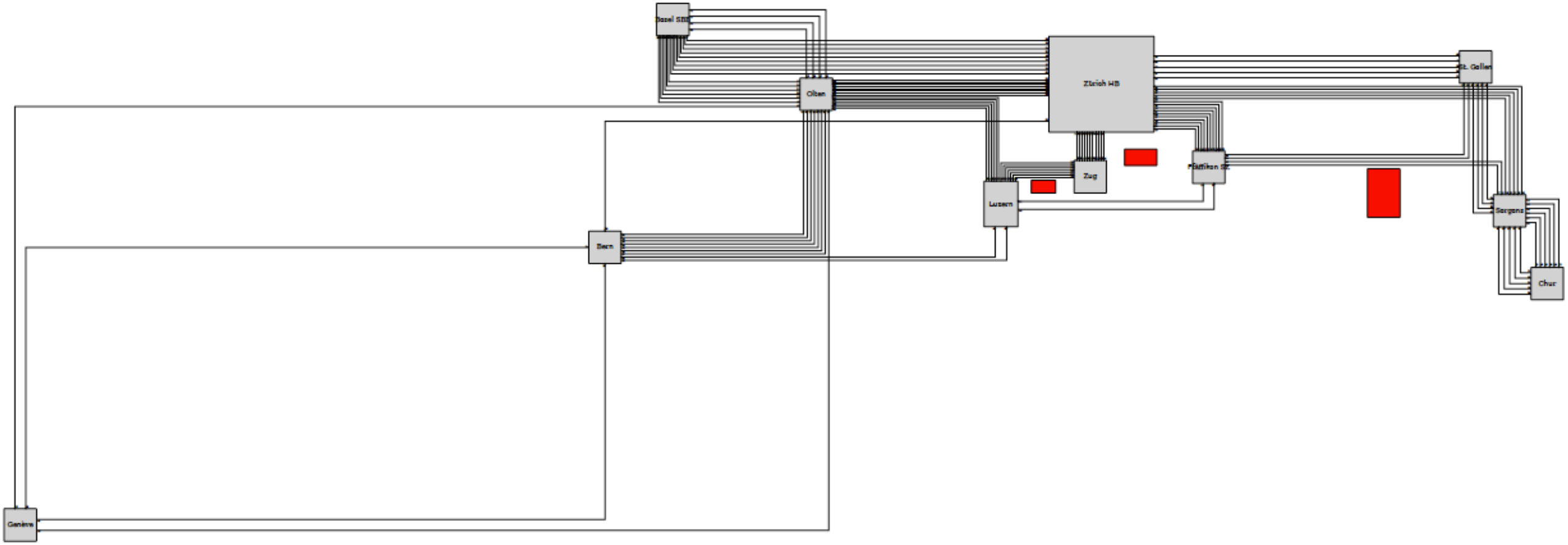
04) Graphviz cont.



Edge overlapping problem

04) Graphviz cont.

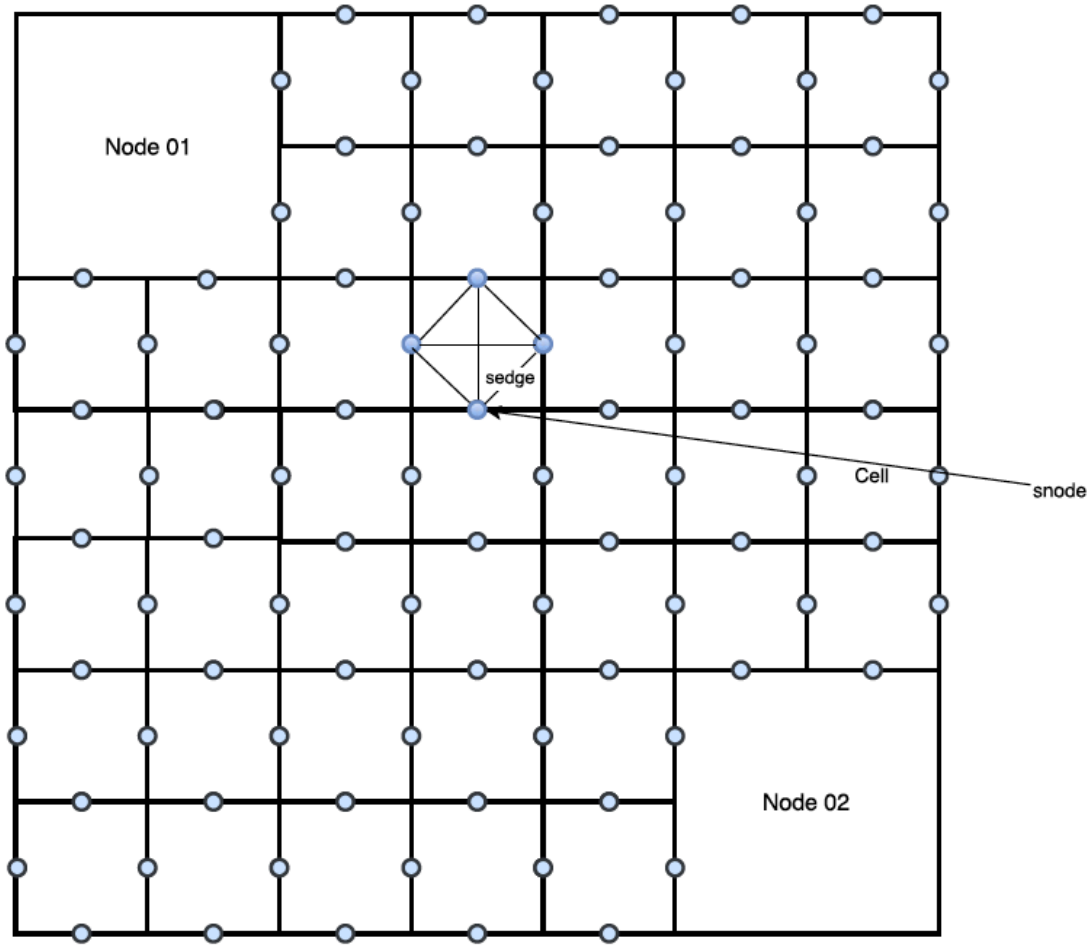
- Prevent overlapping of parallel edges
  - 01) Using dummy nodes



Frequency graph with dummy nodes

04) Graphviz cont.

02) Modify the weight increasing mechanism of edges



Maze object create by Graphviz

04) Graphviz cont.

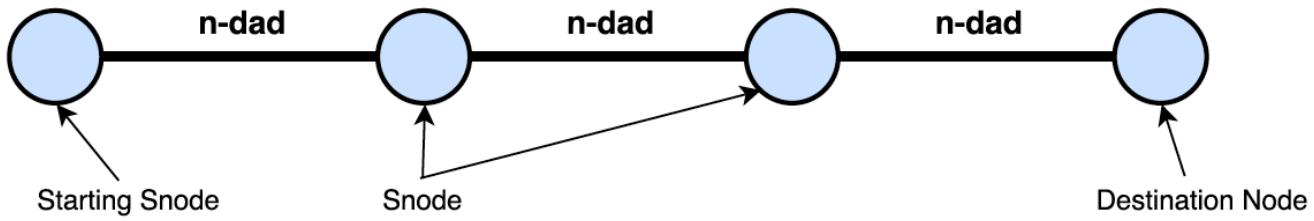
Edge drawing mechanism

1) Collects all the out edges of nodes in the graph into a collection

2) For each out edge, create two snodes (sn and dn) which correspond to starting cell and destination cell

3) Then finds the shortest path between sn and dn using the Dijkstra algorithm

4) Shortest path is stored by storing the reference to the next snode via n_dad attribute and storing sedges in sedge attribute of snode



5) Once all the lines are routed as shortest paths, then the graph drawing starts and completes the drawing of the graph

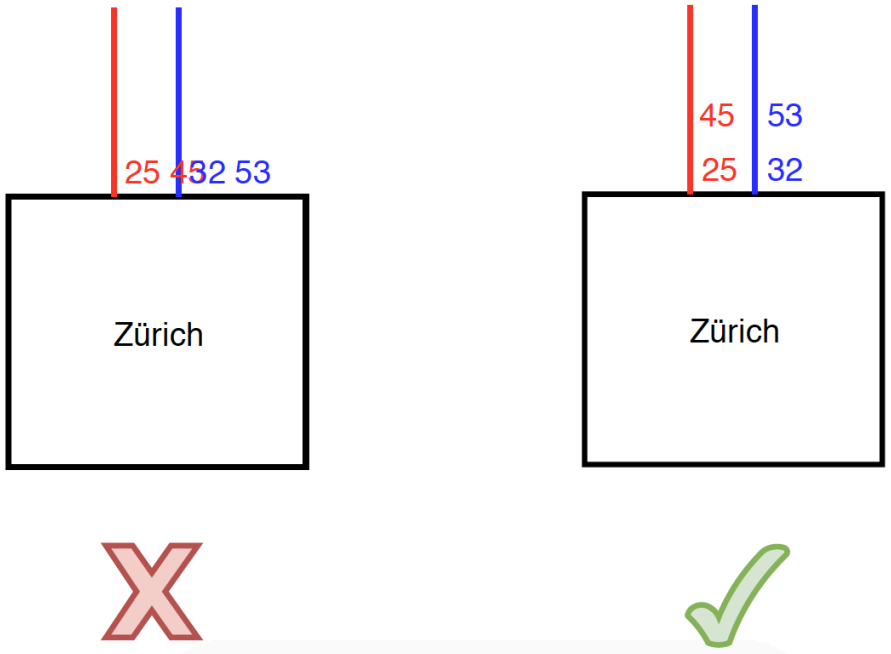
04) Graphviz cont.

Modification of UpdateWt() function

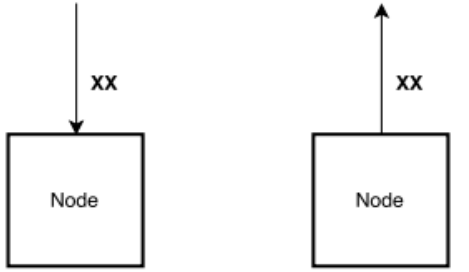
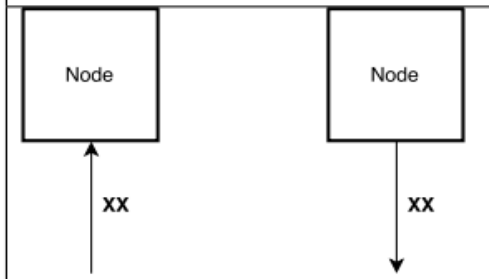
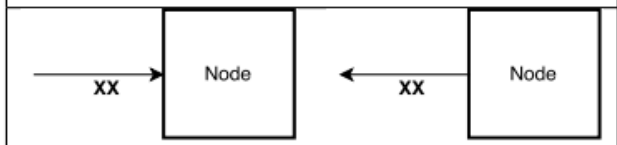
```
static void updateWt (cell* cp, sedge* ep, int sz)
{
   ep->cnt++;
   int x = 10;
   double exponentValue = exp(((double)1/sz) * 200) ;

   double alwdPaths = (double)sz / x;
   double costForOnePath = ((double)BIG / alwdPaths) * exponentValue;
   costForOnePath = (costForOnePath > BIG) ? BIG :costForOnePath;


   ep->weight += costForOnePath;

   // This was the default version of updateWt function
   if (ep->cnt > sz) {
      ep->cnt = 0;
      ep->weight += BIG;
   }
}
```

04) Graphviz cont.

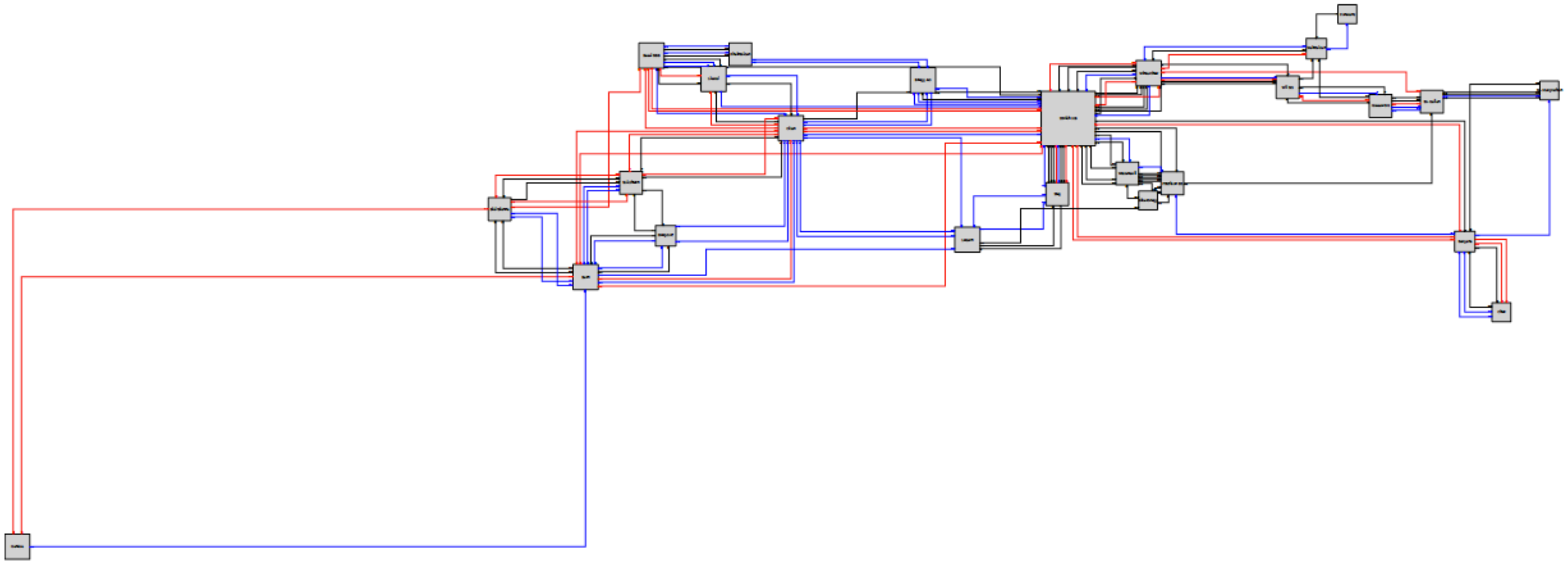
Modification for creating the bidirectional edges



Rendering of two values in edge label

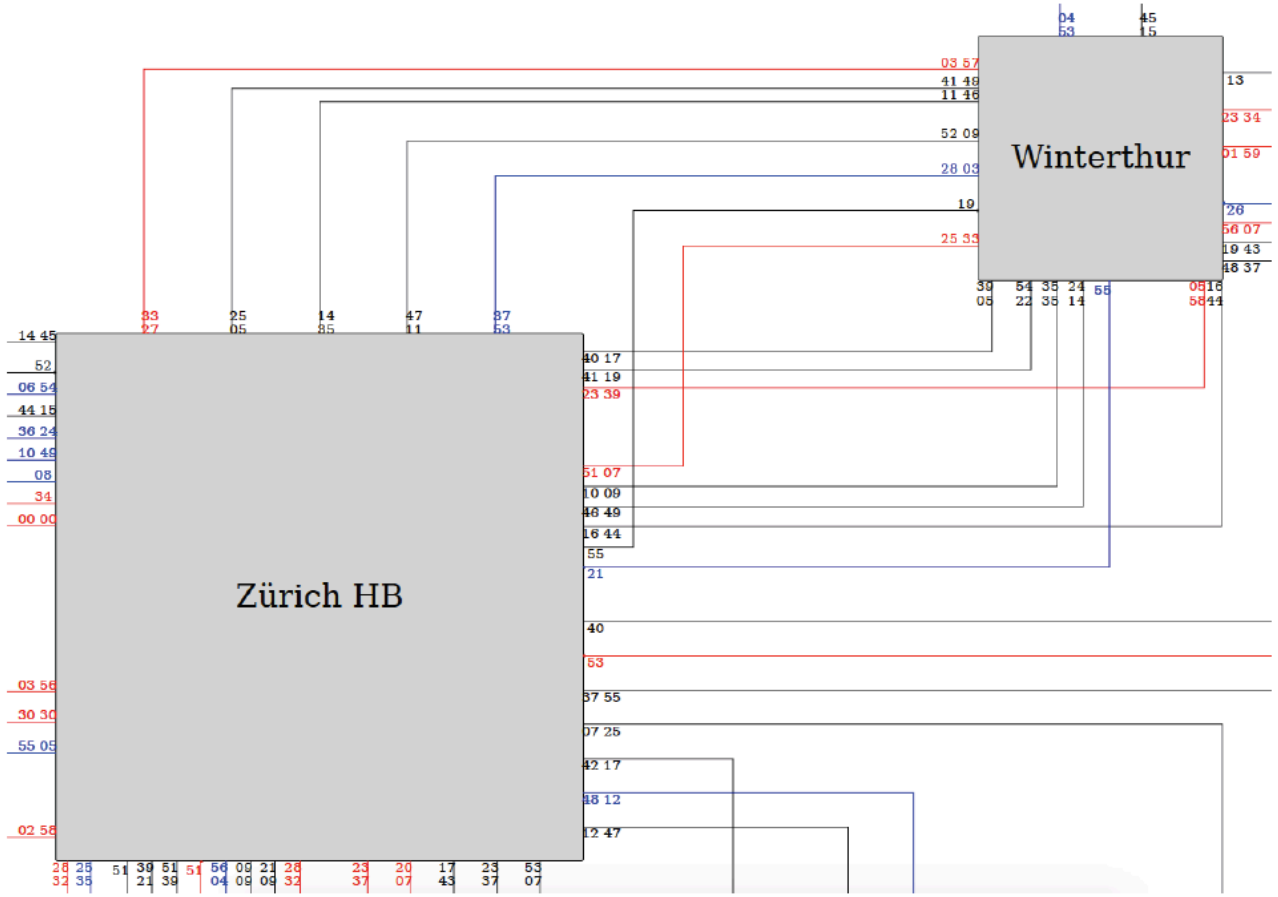
Different ways of connecting orthogonal edges with node	Angle between orthogonal edge and the node
	1.13446
	-0.43633
	-2.00712
	2.70526

04) Graphviz cont.



Frequency graph with bidirectional edges

04) Graphviz cont.



Bidirectional edges between Zürich HB and Winterthur

Evaluation

Time required for frequency finding algorithm and human-friendly frequency finding algorithm

- Configurations: Intel(R) Xeon(R) CPU E5640 @ 2.67GHz, 65 GB

Round	Time taken in milliseconds
1	11096
2	11095
3	11124
4	11051
5	11110

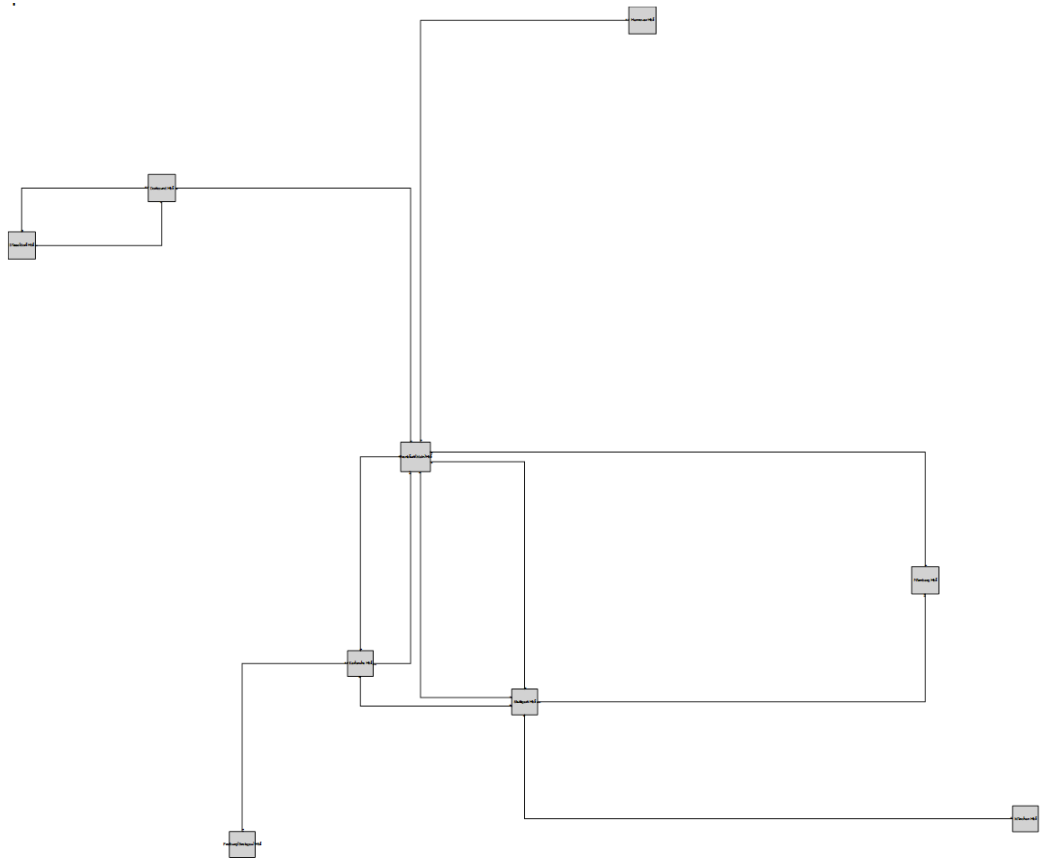
Time taken for frequency finding algorithm

Round	Time taken in milliseconds
1	9516
2	9636
3	9405
4	9243
5	9281

Time taken for human-friendly frequency finding algorithm

Evaluation cont.

Evaluation of the frequency graph rendered on Swiss Railway GTFS data and Deutsche Bahn GTFS data



Frequency graph rendered on Deutsche Bahn GTFS data

Evaluation cont.

Evaluation on the trips covered by the frequency coverages out of the total trips

Start and stop station	Total number of departure times	Total number of departure times covered by the frequency covers	Percentage
Zurich HBF - Olten	68	63	92.64%
Zurich HBF - Basel SBB	77	77	100%
Olten - Bern	84	76	90.47%
Geneve - Luzern	15	14	93.33%
Sargans - Chur	88	83	94.31%

For the selected nodes, the total number of departure times covered by frequency coverages in the Swiss GTFS dataset

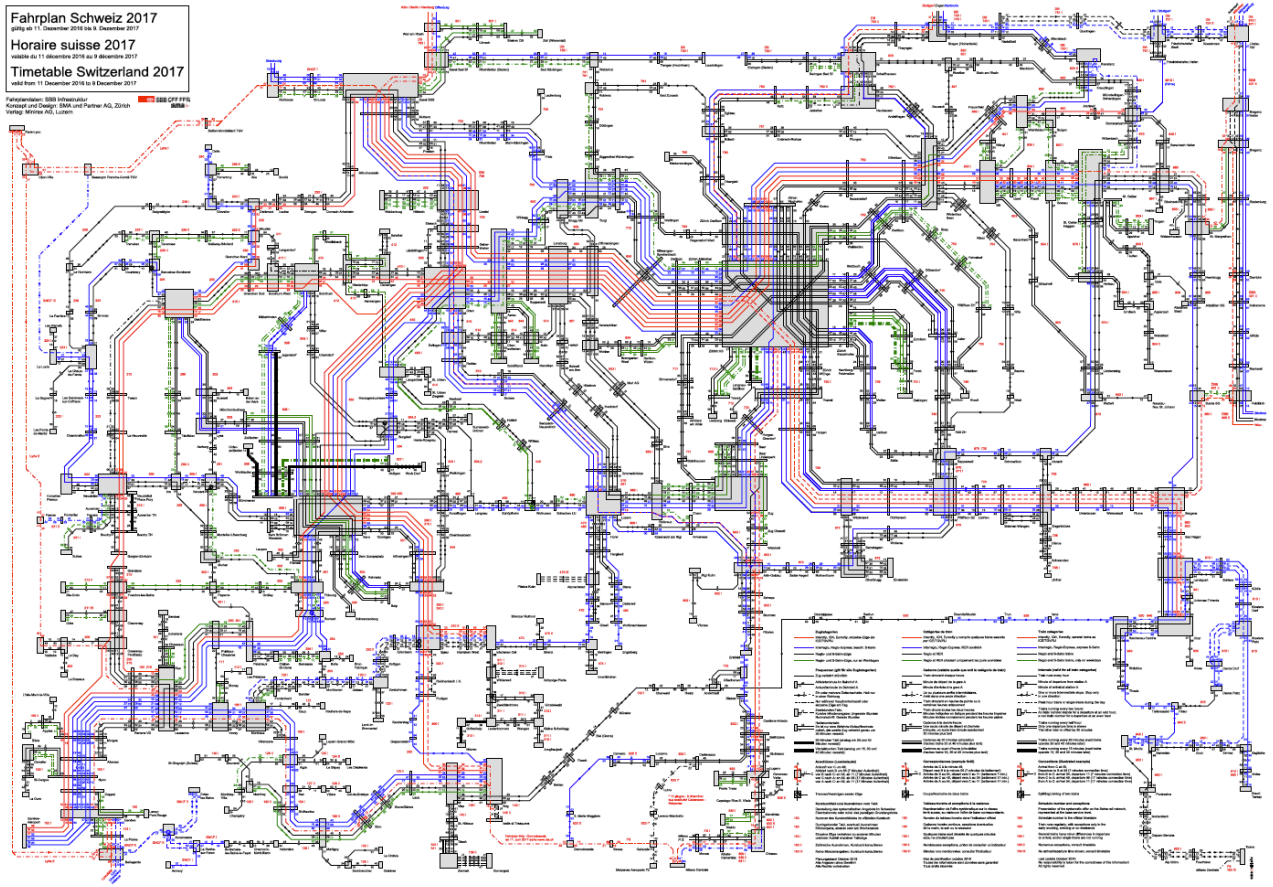
Evaluation cont.

Start and stop station	Total number of departure times	Total number of departure times covered by the frequency covers	Percentage
Freiburg(Breisgau) Hbf - Frankfurt(Main)Hbf	12	6	50%
Karlsruhe Hbf - Stuttgart Hbf	20	8	40%
Frankfurt(Main)Hbf - Munchen Hbf	9	6	66.66%
Stuttgart Hbf - Nuremberg Hbf	7	7	100%
Dusseldorf HBF - Stuttgart HBF	11	0	0%

For the selected nodes, the total numbers of departure times covered by the frequency coverages in the Deutsche Bahn GTFS data

Evaluation cont.

Evaluation between the automatically generated frequency graph and the manually created Switzerland Timetable-2017 graph



Switzerland Timetable-2017 Frequency Graph [1]

Evaluation cont.

Advantages of manually created frequency map

- Fewer edge crossings
- In-connections and out-connections of the nodes are routed in consistent manner
- Grouping of frequency lines
  - Ex: 15, 20, minute frequency coverages

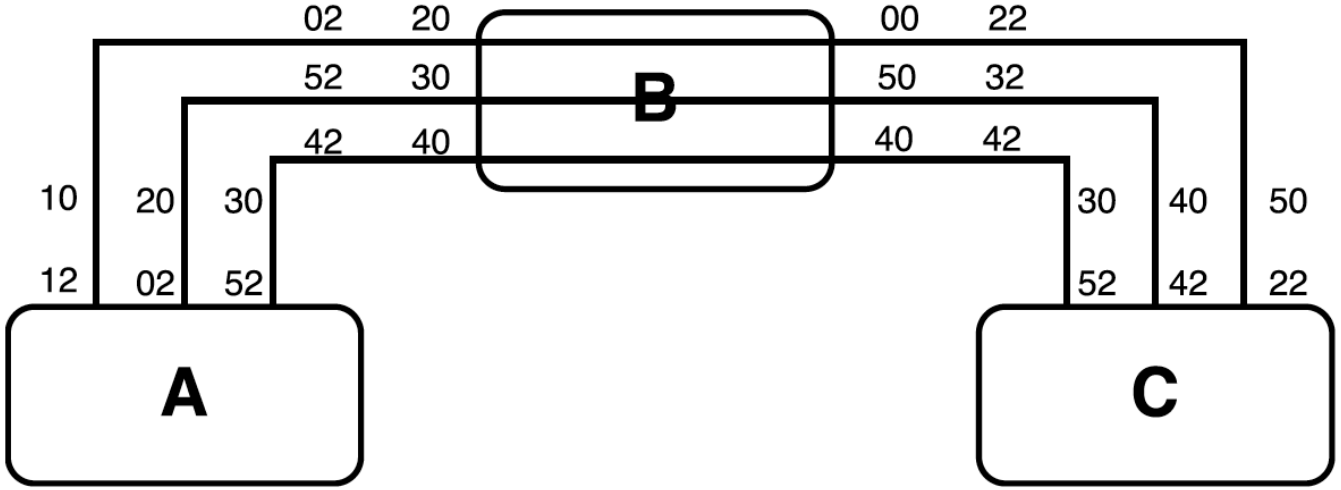
Contribution

- Developed a tool to extract the frequency graph as GeoJSON and Dot language from arbitrary GTFS data
- Experimented with 4 different approaches to render the frequency graph in a nice way
- Reverse-engineered the method used by the ortho layout of NEATO engine and extended it to better handle multigraphs (with many edges between two nodes)
- Implemented multiline edge-label rendering to better support frequency maps
- Evaluated our entire pipeline on the complete rail network of Switzerland and the long distance network of Germany

Future work

- Makes the frequency finding algorithm tolerance for deviations in departures.
  - Ex Karlsruhe HBF the departure times from Freiburg HBF as follows
    - 8:57, 9:56, 10:57, 11:56, 12:57, 13:57 ..

- Connects in connections of a node to its out connection



In and out connections are connected

Thank You for your attention!

Citations

1. Switzerland Timetable-2017 (Frequency Map). http://www.bahnonline.ch/bo/18755/netzgrafik-fahrplan-schweiz-2017.htm.

2. General Transit Feed Specification https://developers.google.com/transit/gtfs/reference

3. geOps, “Public Transporation Feed for Switzerland.” http://gtfs.geops.ch.

Q&A Backup slides

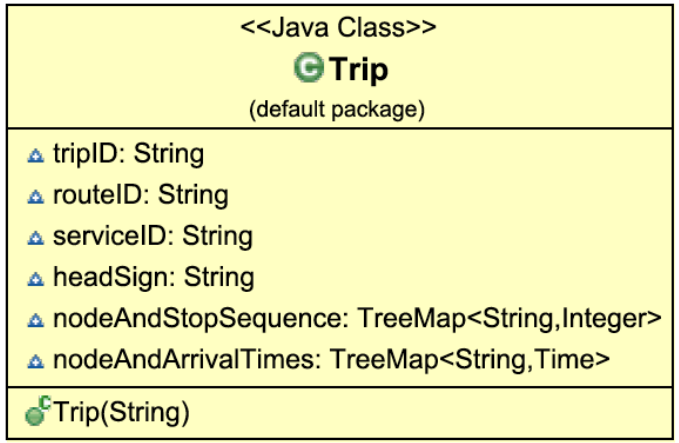
- Data Model

```
route_id,service_id,trip_id,trip_headsign,trip_short_name,direction_id,block_id,shape_id,bikes_allowed,attributes_ch
03002.06____:3002,1:1:s,1,Neckarbischofsheim Nord,3002,,,,0,TS
03003.06____:3003,2:1:s,2,Untergimpern,3003,,,,0,TS
03005.06____:3005,3:1:s,3,Hüffenhardt,3005,,,,0,TS
03006.06____:3006,4:1:s,4,Neckarbischofsheim Nord,3006,,,,0,TS
03007.06____:3007,5:1:s,5,Hüffenhardt,3007,,,,0,TS
03008.06____:3008,6:1:s,6,Neckarbischofsheim Nord,3008,,,,0,TS
03009.06____:3009,7:1:s,7,Hüffenhardt,3009,,,,0,TS
03012.06____:3012,091831,8,Neckarbischofsheim Nord,3012,,,,0,
03013.06____:3013,091831,9,Hüffenhardt,3013,,,,0,
03014.06____:3014,091831,10,Neckarbischofsheim Nord,3014,,,,0,
03015.06____:3015,091831,11,Hüffenhardt,3015,,,,0,
03016.06____:3016,091831,12,Neckarbischofsheim Nord,3016,,,,0,
```

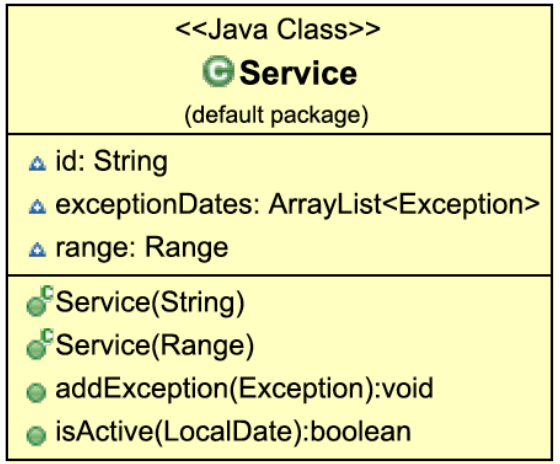
trips.txt

```
service_id,date,exception_type
10704:4:4:s,20171202,1
25358:1:s,20170626,1
25358:1:s,20170627,1
25358:1:s,20170625,1
25358:1:s,20171211,1
25358:1:s,20171210,1
25358:1:s,20170628,1
25358:1:s,20170629,1
25358:1:s,20170703,1
25358:1:s,20170702,1
```

calendar_dates.txt



Trip Class



Service Class

Q&A Backup slides

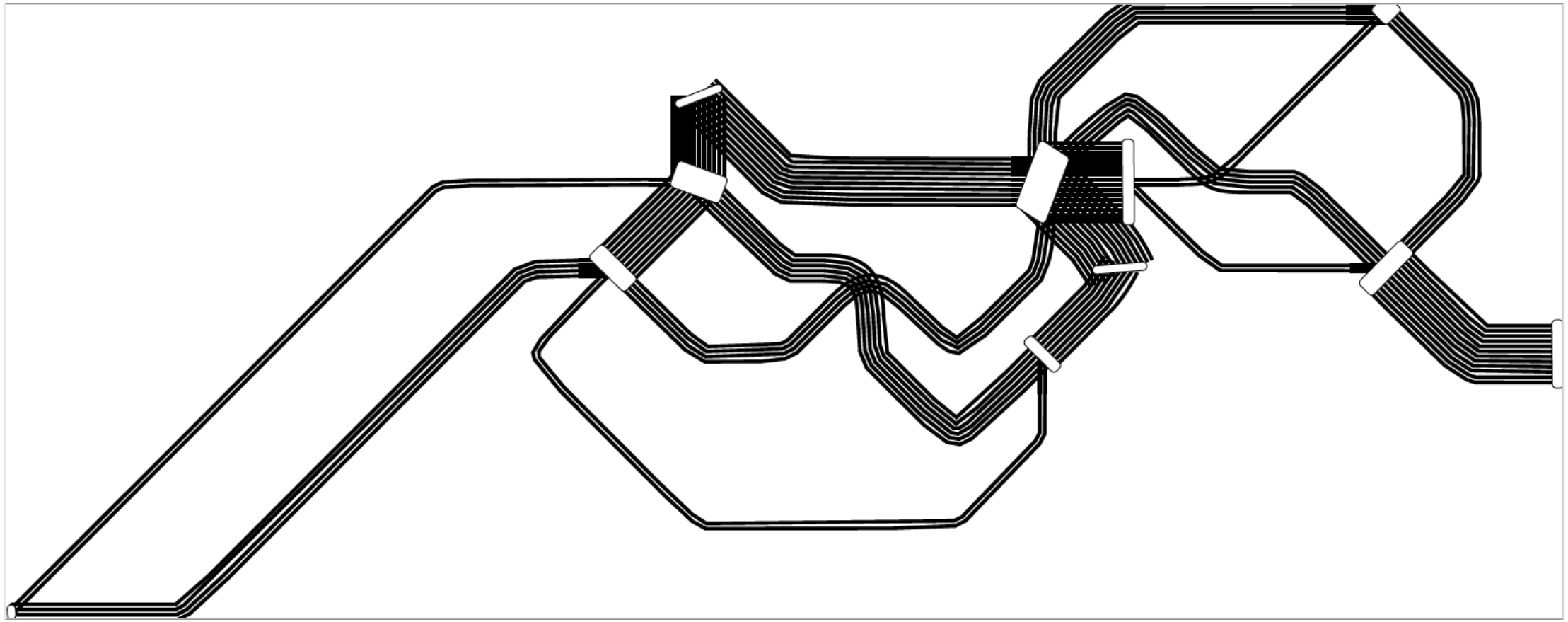
Extraction of frequency coverage between interesting nodes cnt.

```
1  public ArrayList<String> searchTripsBetweenNodes(Node nodeA, Node nodeB) {
2     Set<String> intersection = new HashSet<String>(nodeA.tripIDs);
3     intersection.retainAll(nodeB.tripIDs);
4     ArrayList<String> list = new ArrayList<String>();
5
6     for (String tripID :intersection) {
7        if ((trips.get(tripID).nodeAndStopSequence.get(nodeB.id)
8              - trips.get(tripID).nodeAndStopSequence.get(nodeA.id)) > 0) {
9           list.add(tripID);
10       }
11    }
12    return list;
13 }
```

Algorithm to find the tripIDs which covers Node A and B

Q&A Backup slides

Octi Tool



Swiss frequency graph rendered by Octi