Public-Transit Data Extraction From OpenStreetMap Data

Zhiwei Zhang

February 13, 2017

Reviewer: Prof. Dr. Hannah Bast

Supervisor: Prof. Dr. Hannah Bast

Tutor: Patrick Brosi

Table of Contents

Table of Contents

Introduction

1. OpenStreetMap?

2. Meaning of topic?

Introduction

1. OpenStreetMap?
   an open-source project providing free geographic data

2. Meaning of topic?

Introduction

1. OpenStreetMap?
   an open-source project providing free geographic data

2. Meaning of topic?

Introduction

1. OpenStreetMap?
   an open-source project providing free geographic data

2. Meaning of topic?
   Provide data-support for route planing softwares

Table of Contents

Mapping Relation

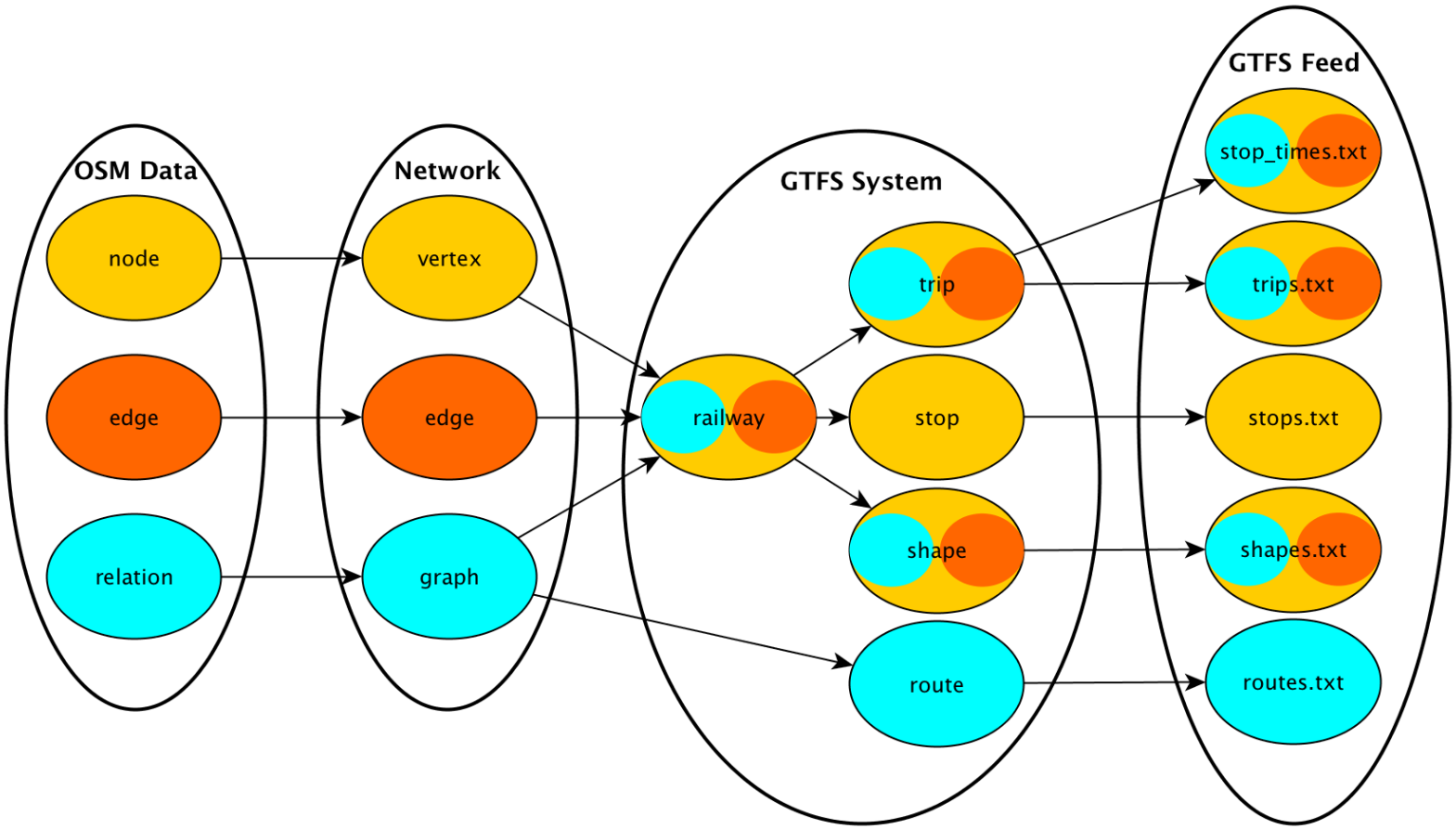
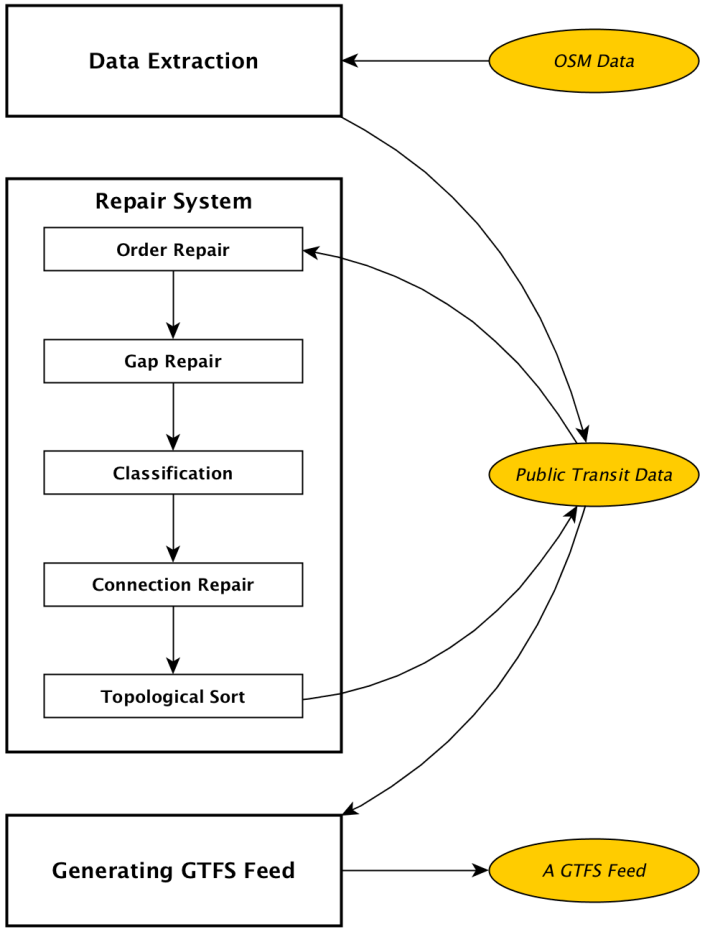
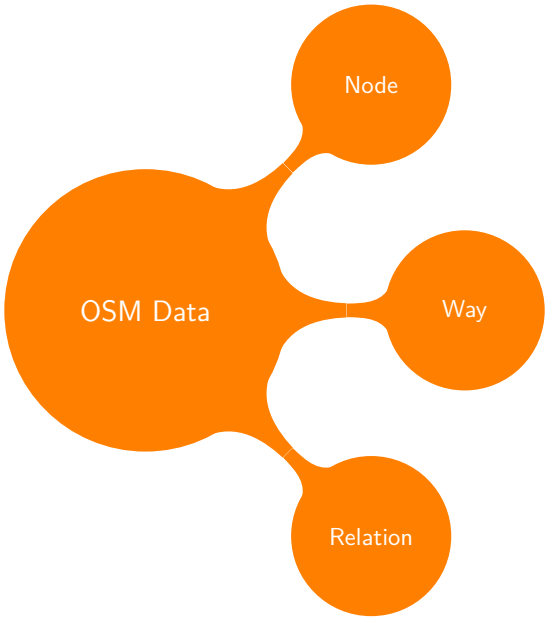


Table of Contents

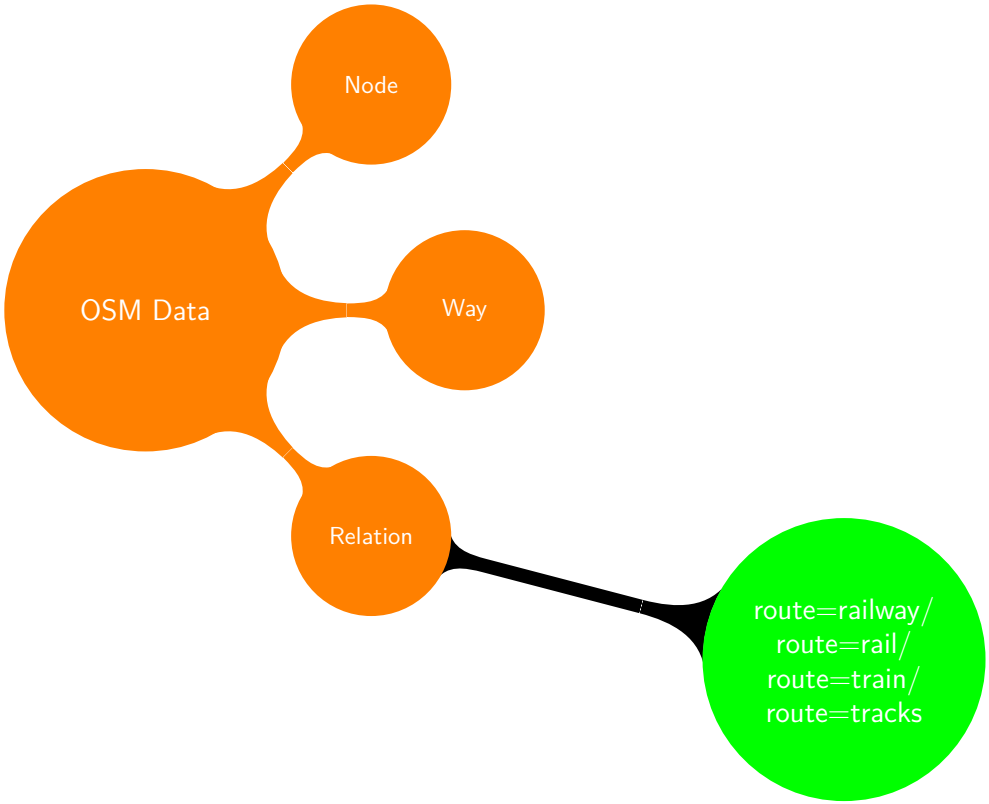
Brief


Data Extraction
OSM Data
Repair System
Order Repair
Gap Repair
Classification
Connection Repair
Topological Sort
Public Transit Data
Generating GTFS Feed
A GTFS Feed

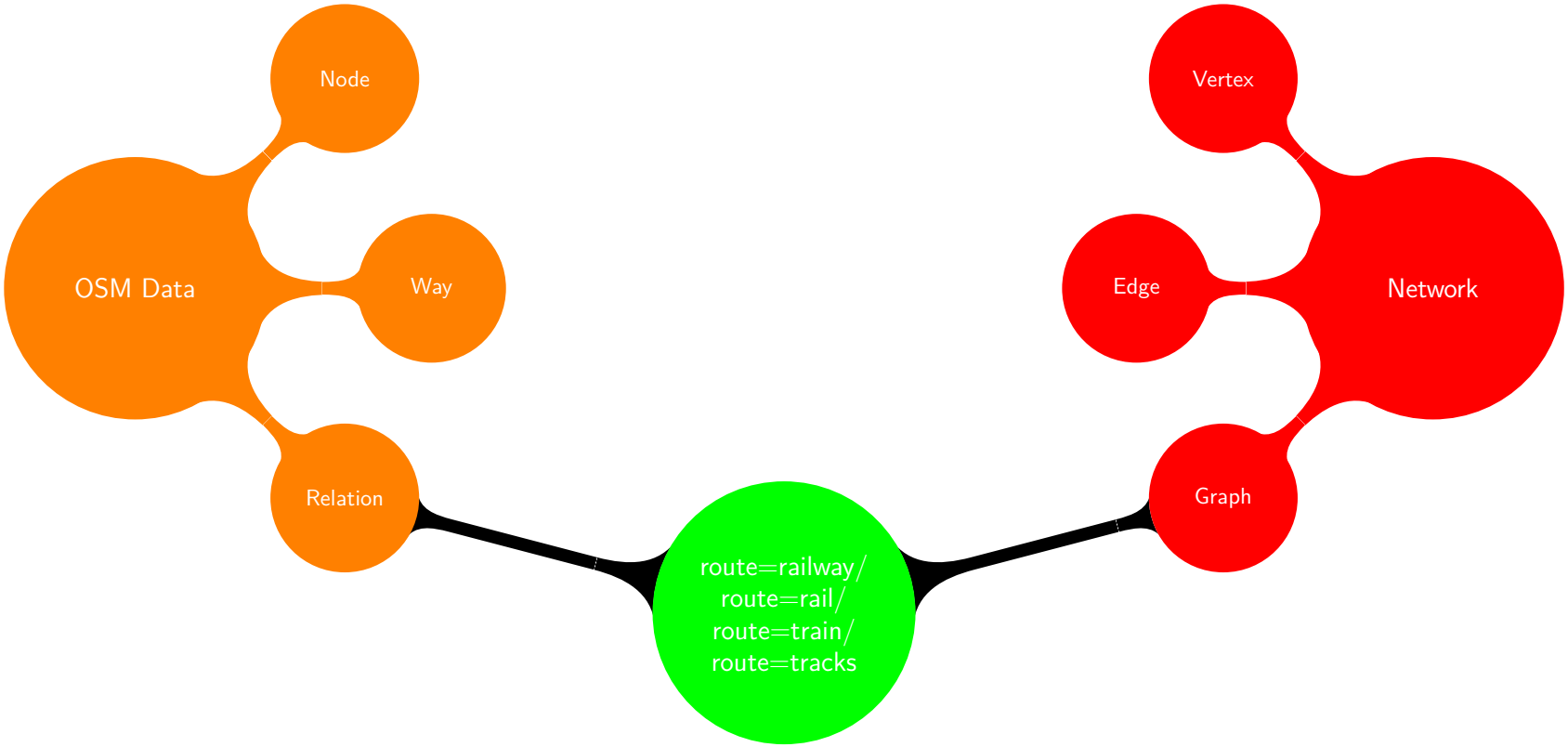
Data Extraction



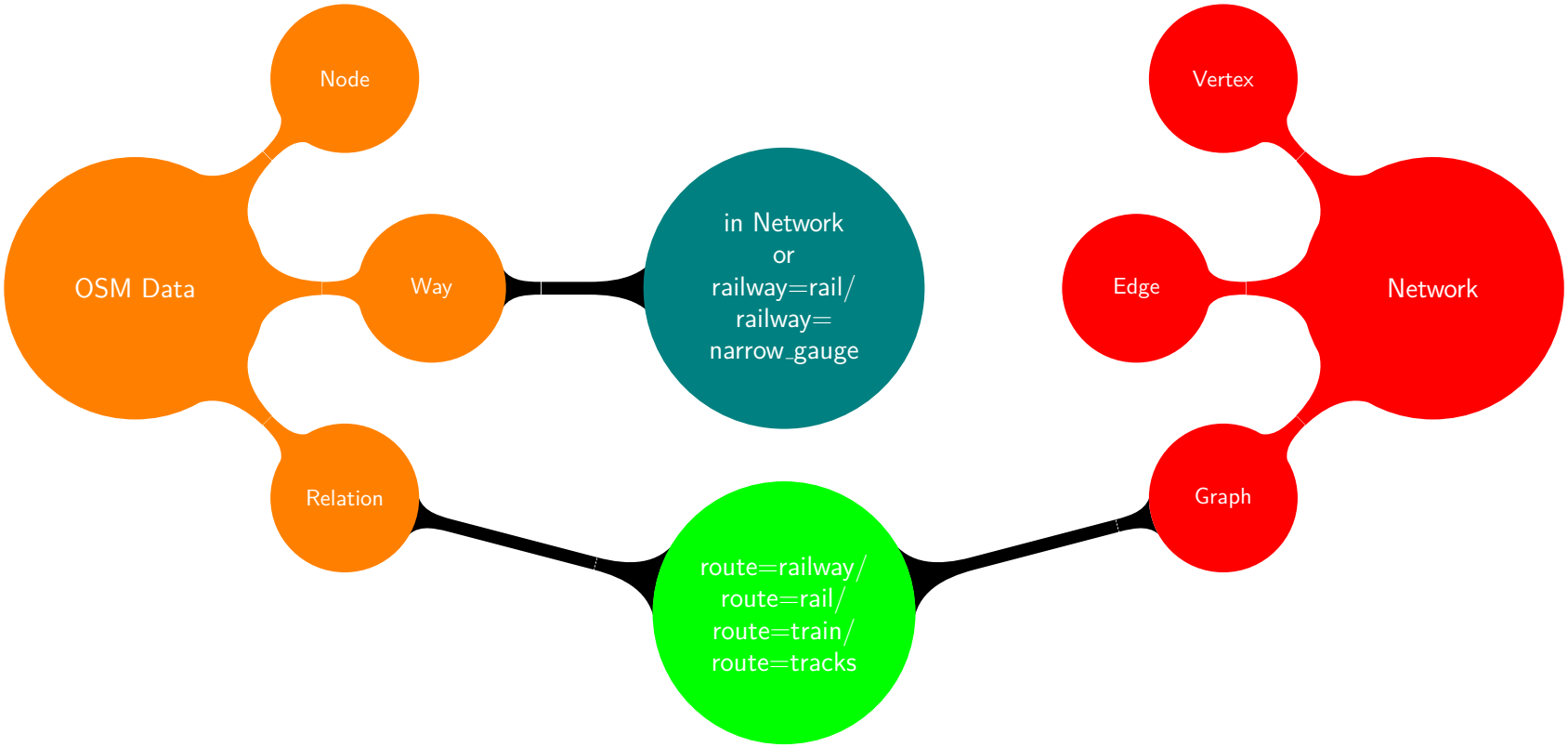
Data Extraction



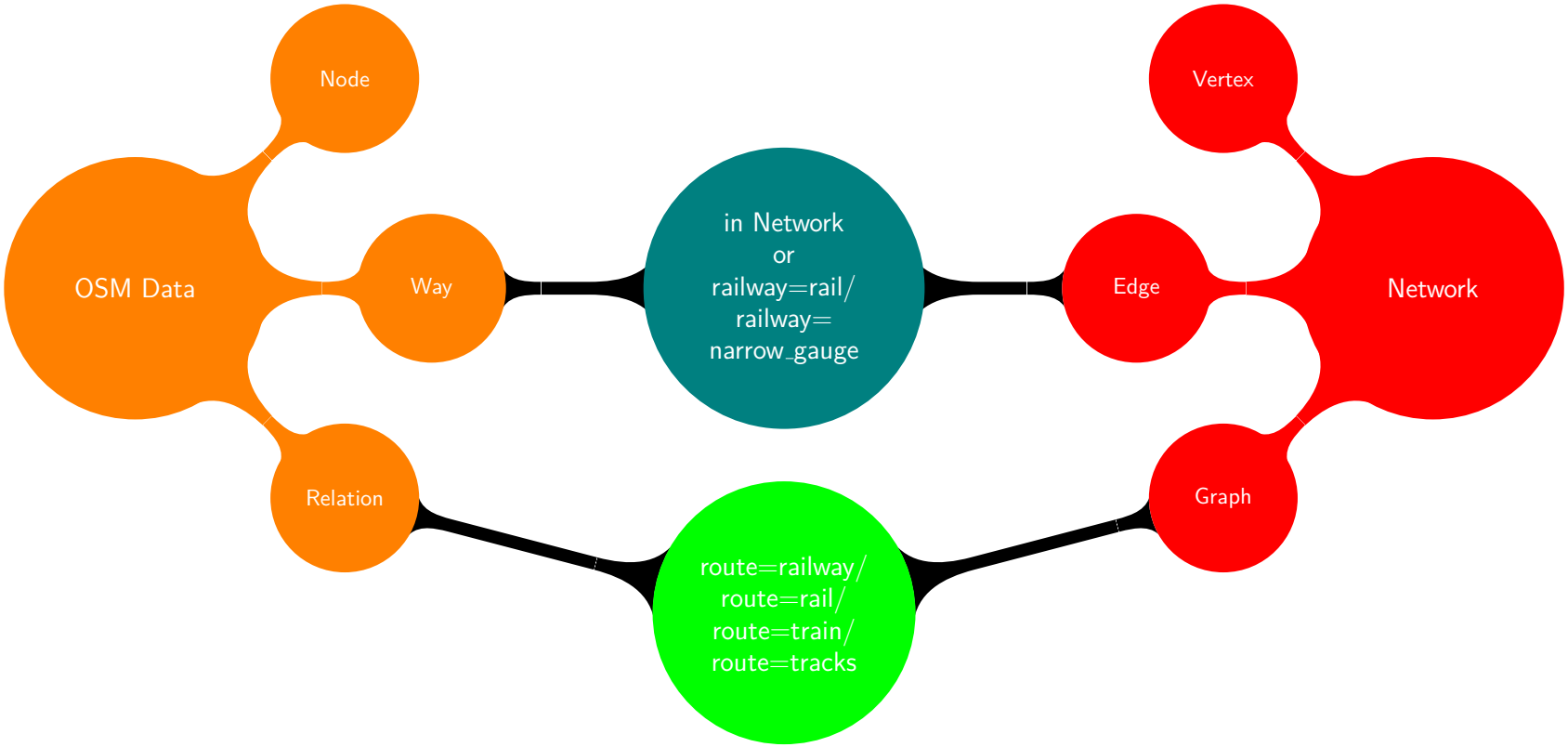
Data Extraction



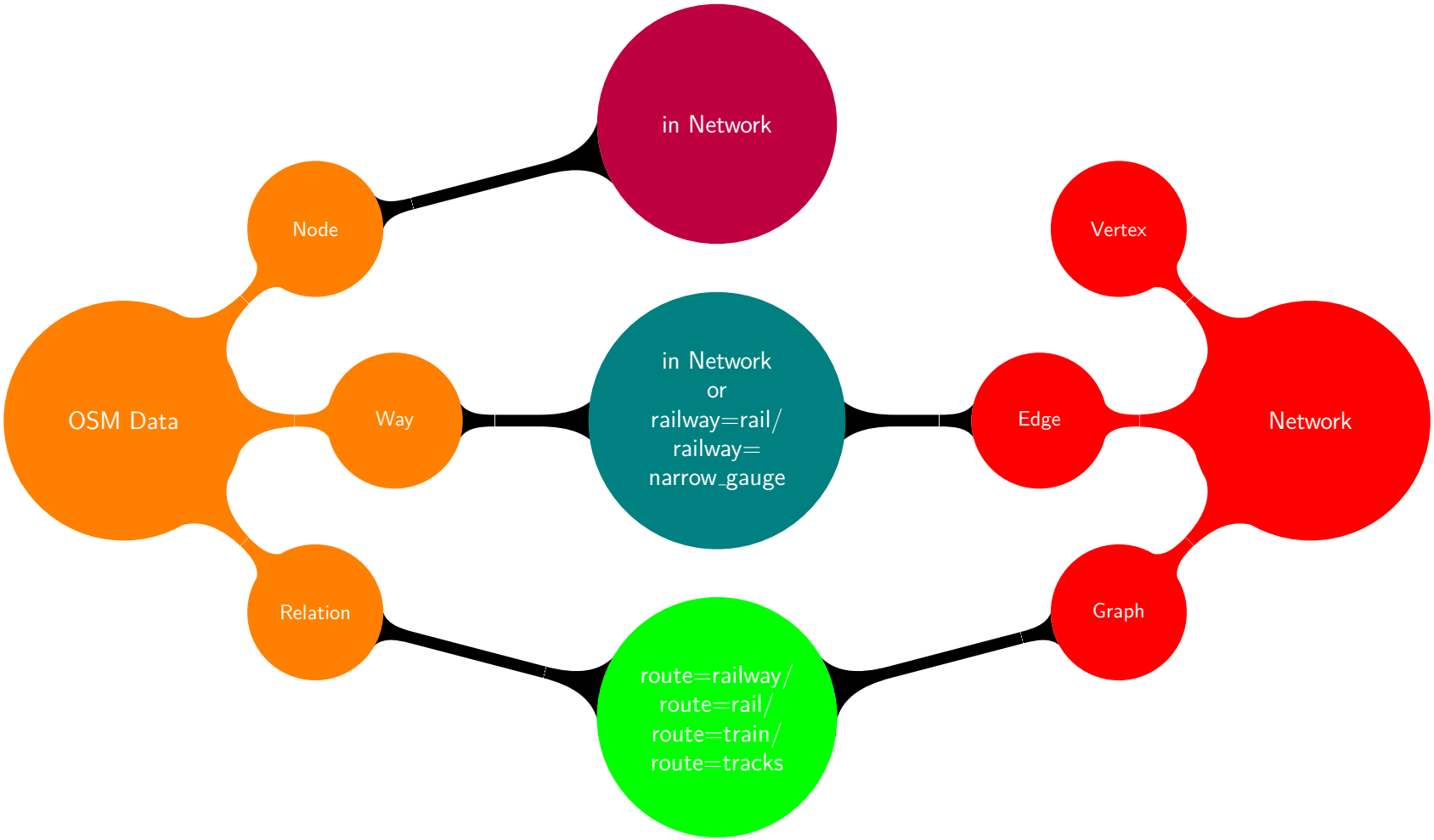
Data Extraction



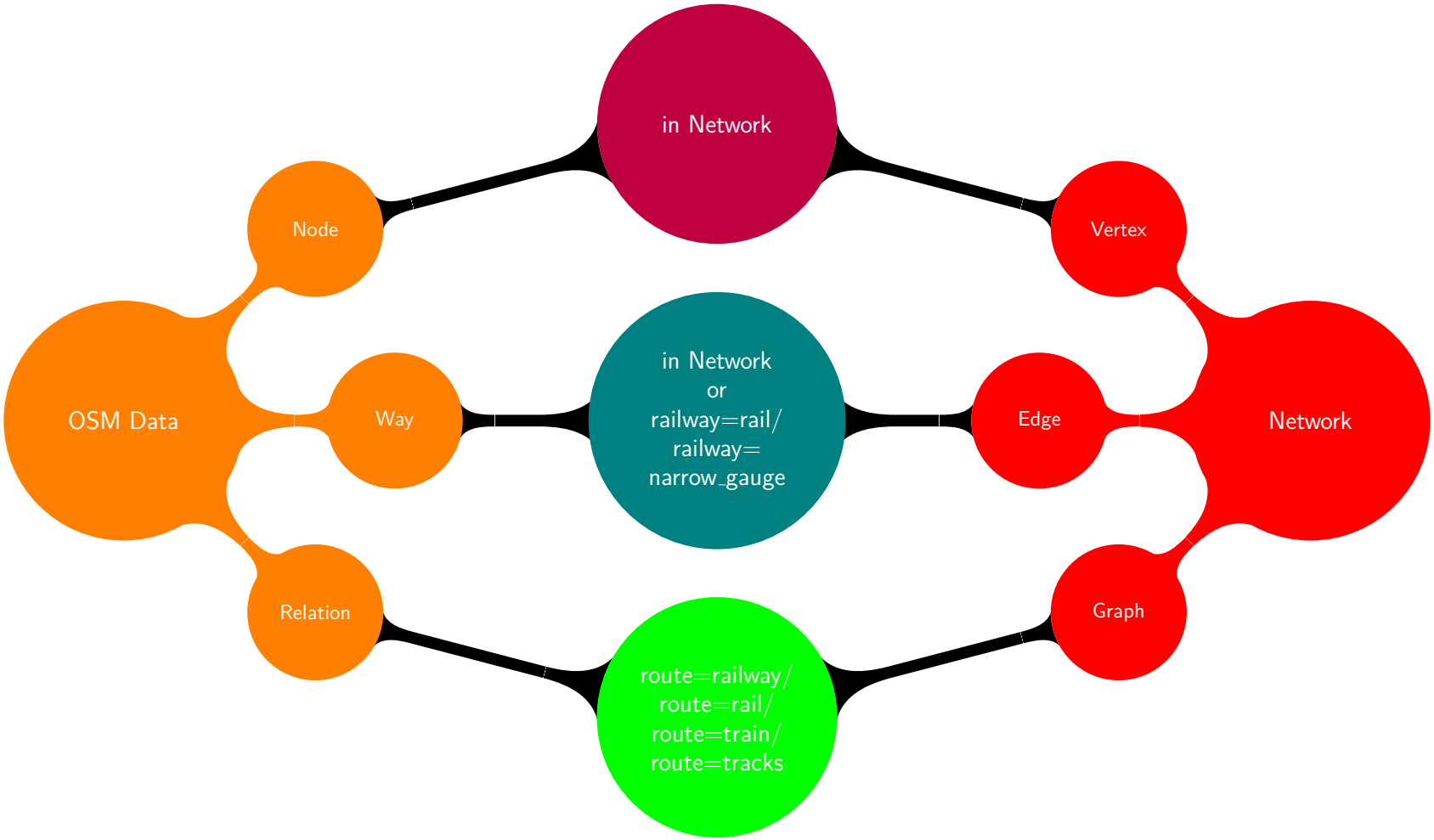
Data Extraction



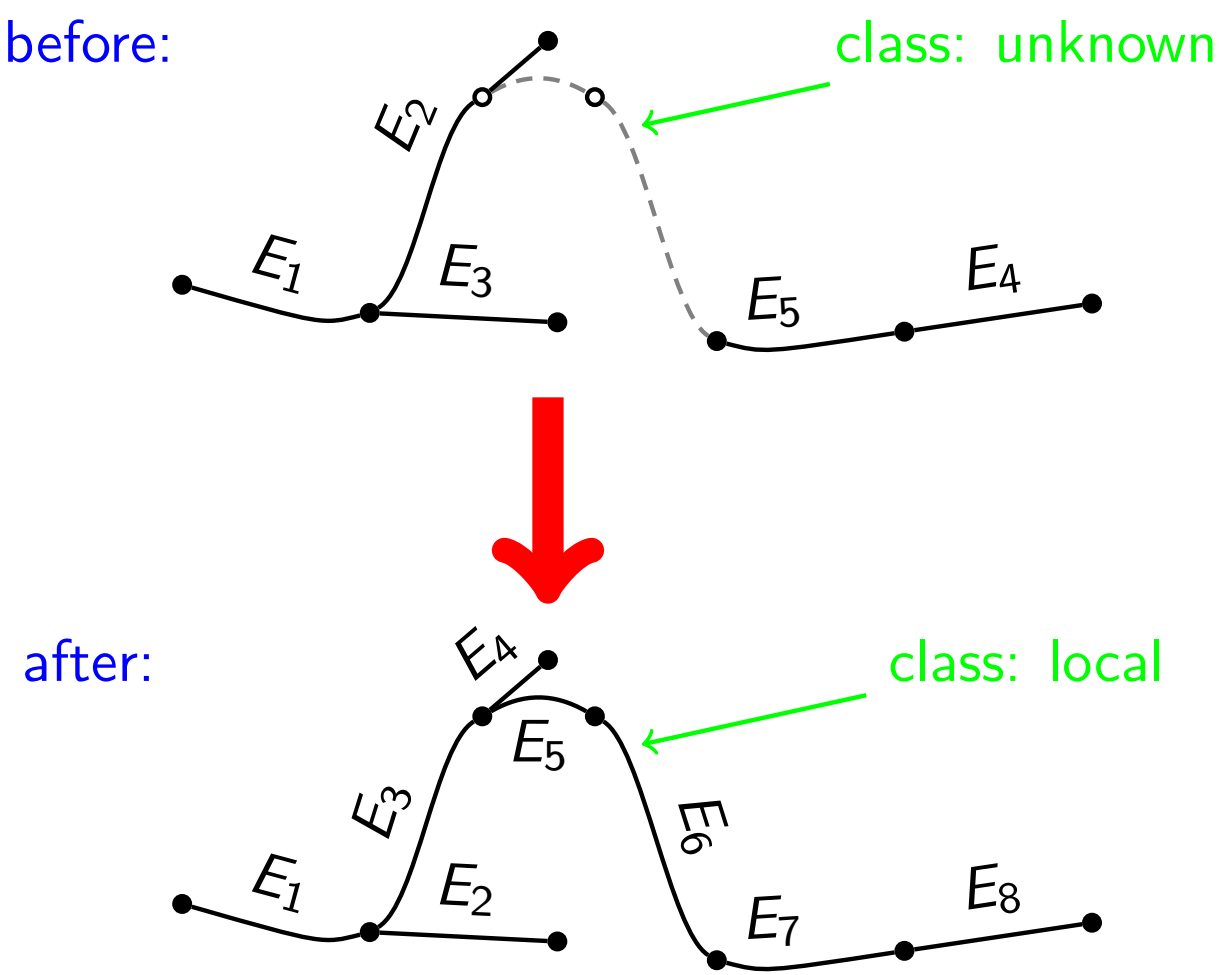
Data Extraction



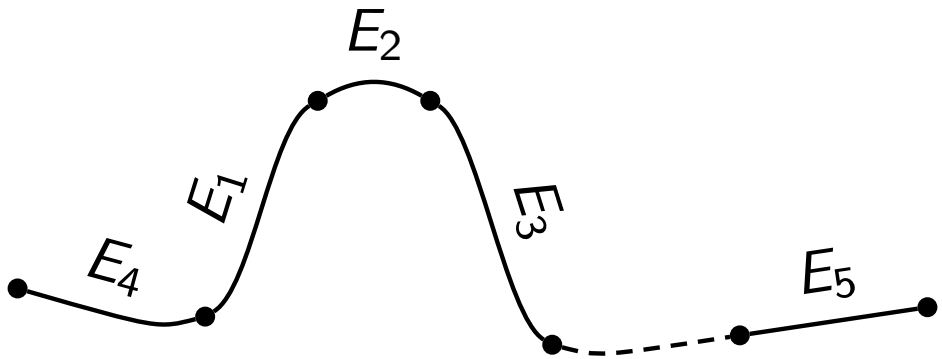
Data Extraction



Repair System



Oder Repair - Introduction



—— : contained in this graph

--- : not contained in this graph

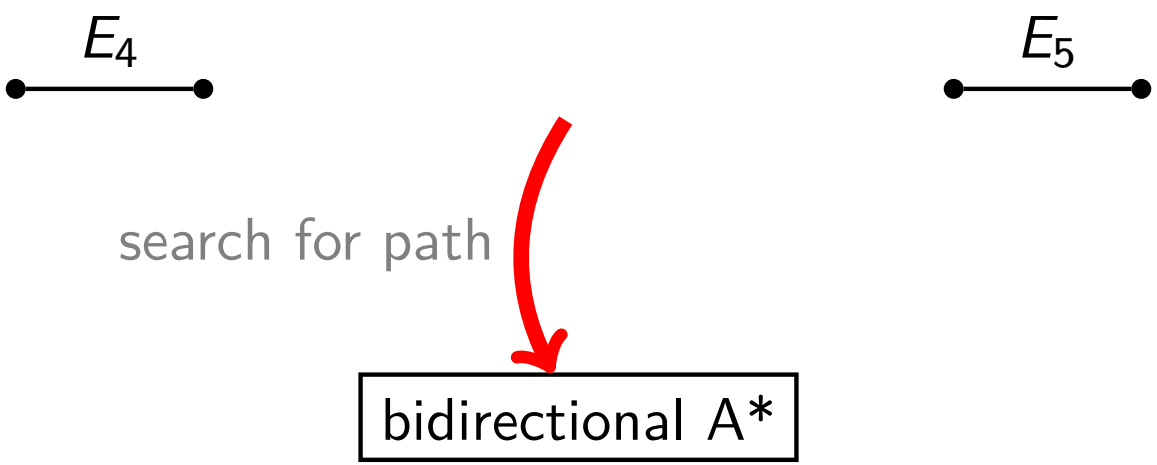
Original Edge Order (containing fake gaps):

$$E_1 \rightarrow E_2 \rightarrow E_3 \dashrightarrow E_4 \dashrightarrow E_5$$

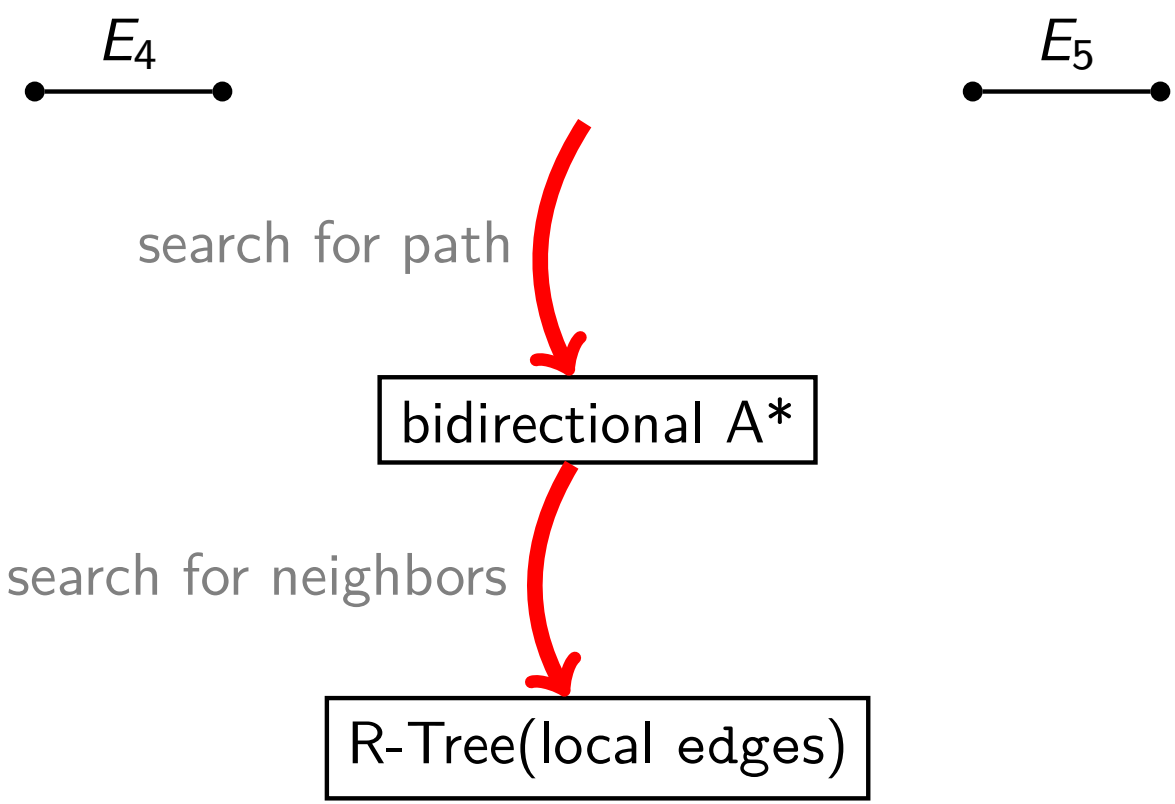
$\dashrightarrow$: means a gap.

Order Repair - fixing gap between E_4 and E_5

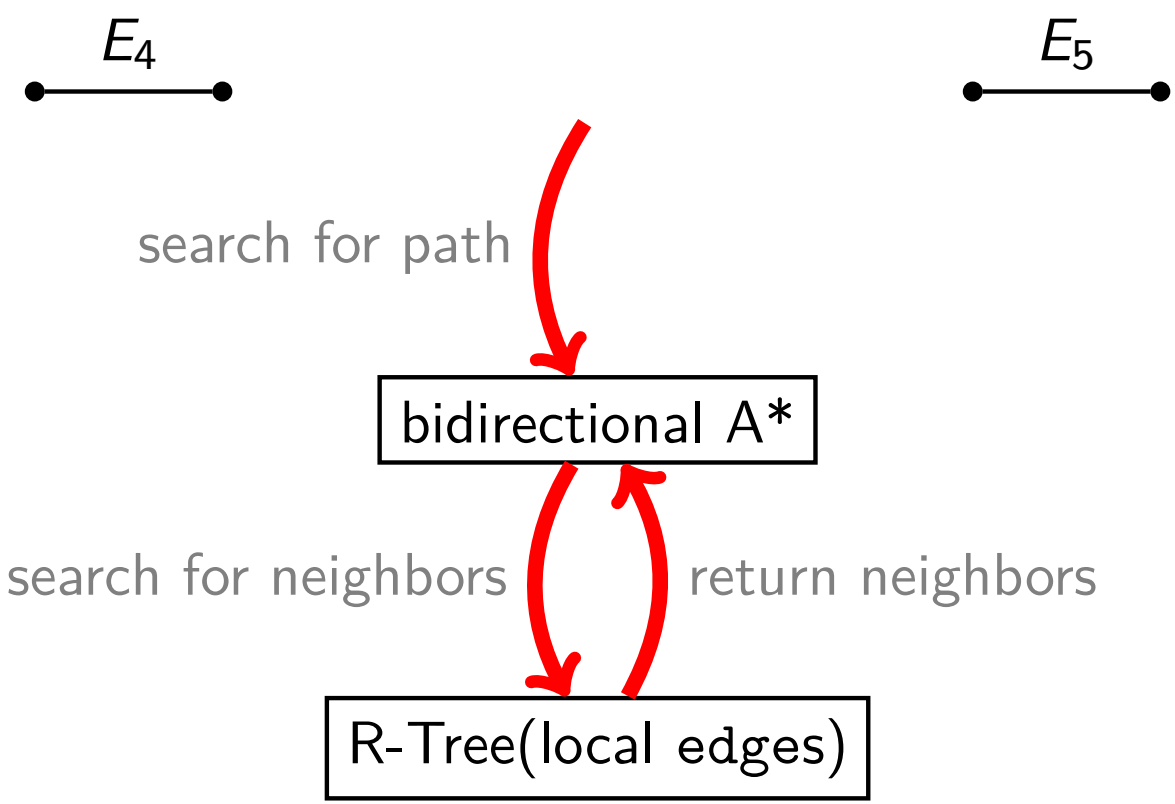
Order Repair - fixing gap between E_4 and E_5



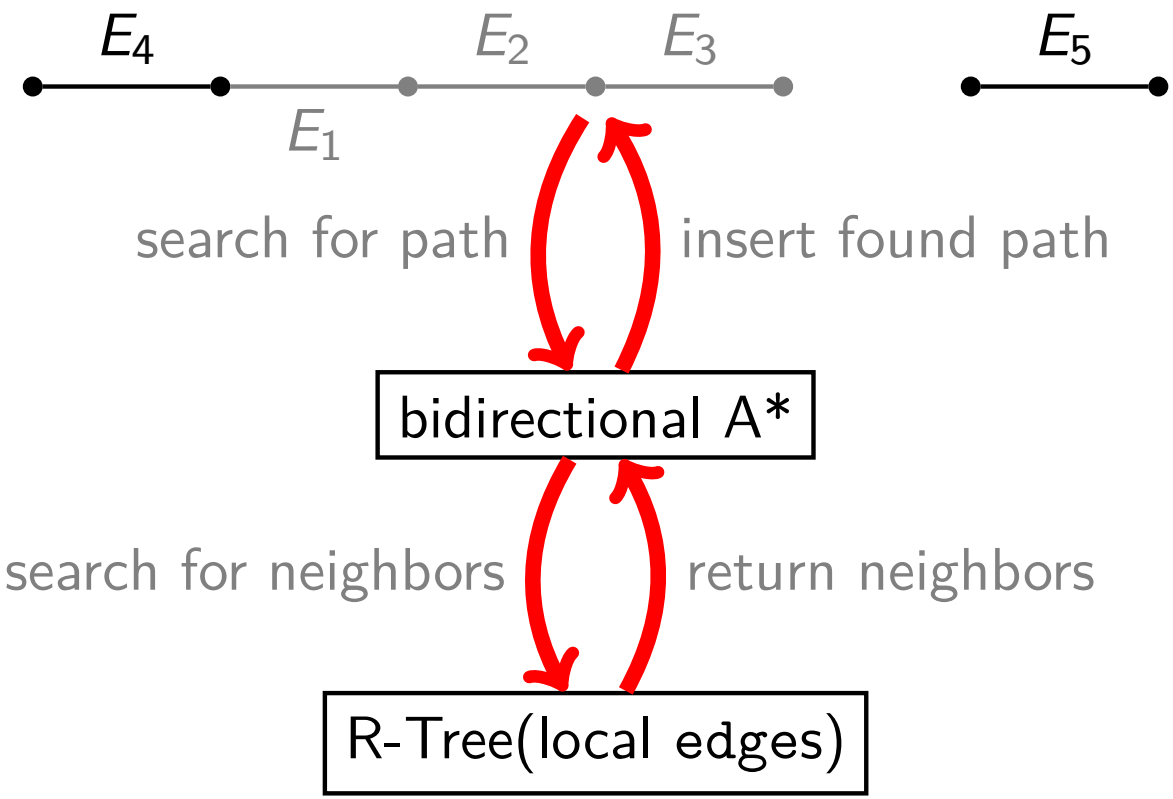
Order Repair - fixing gap between E_4 and E_5



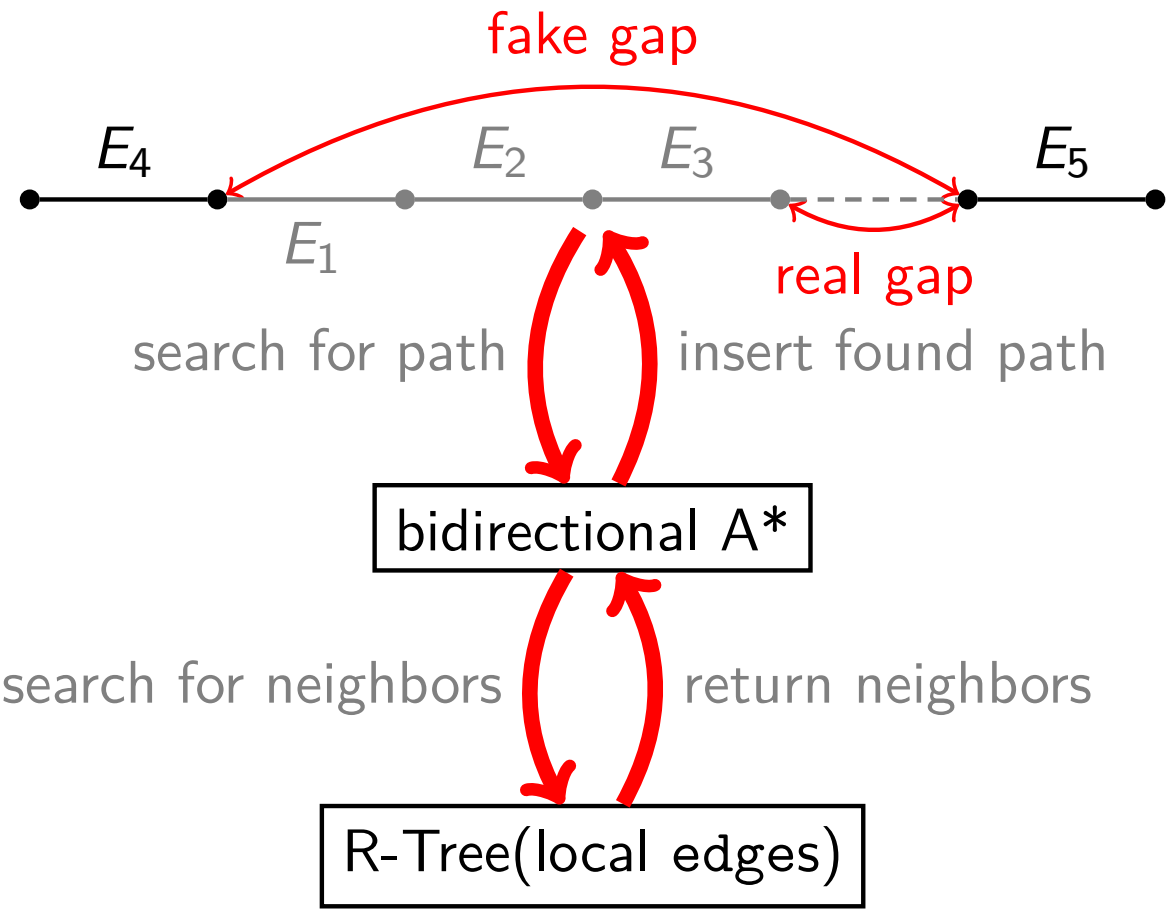
Order Repair - fixing gap between E_4 and E_5



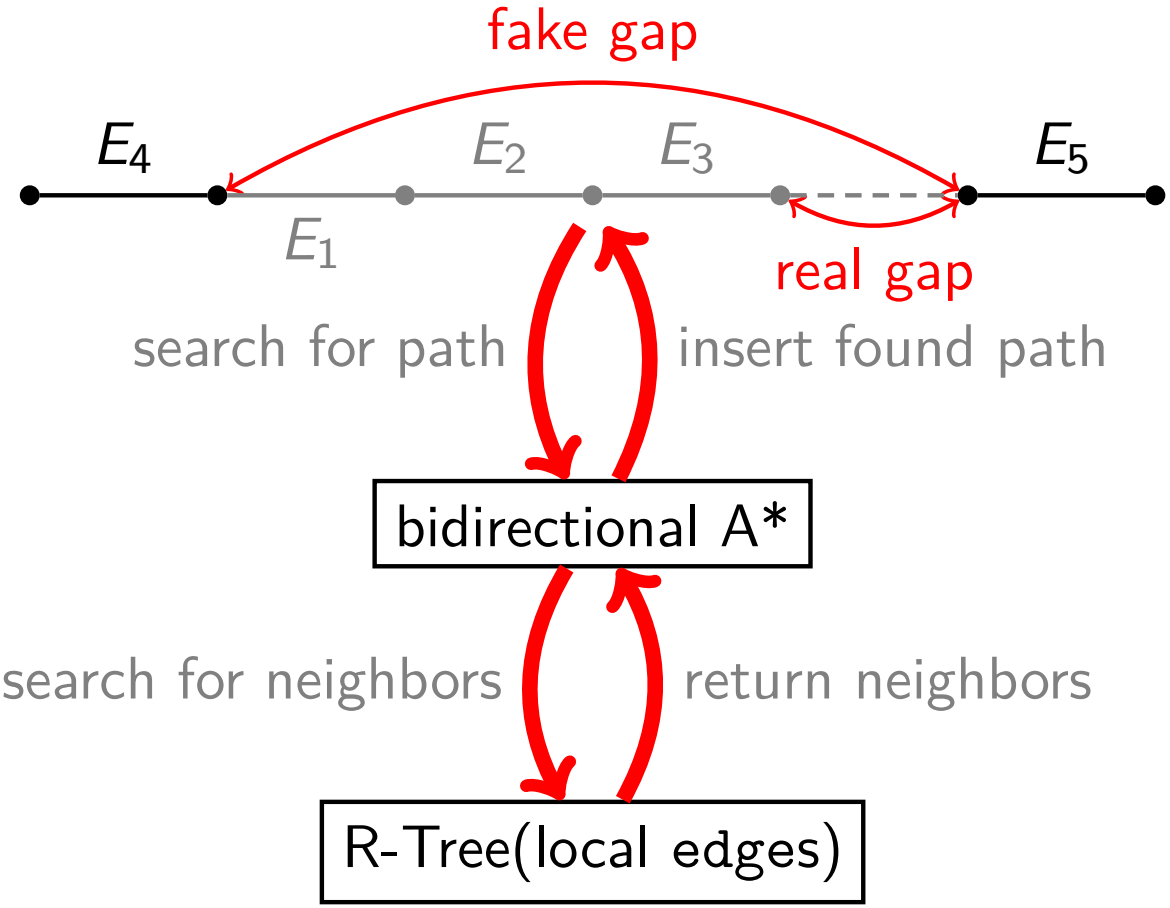
Order Repair - fixing gap between E_4 and E_5



Order Repair - fixing gap between E_4 and E_5



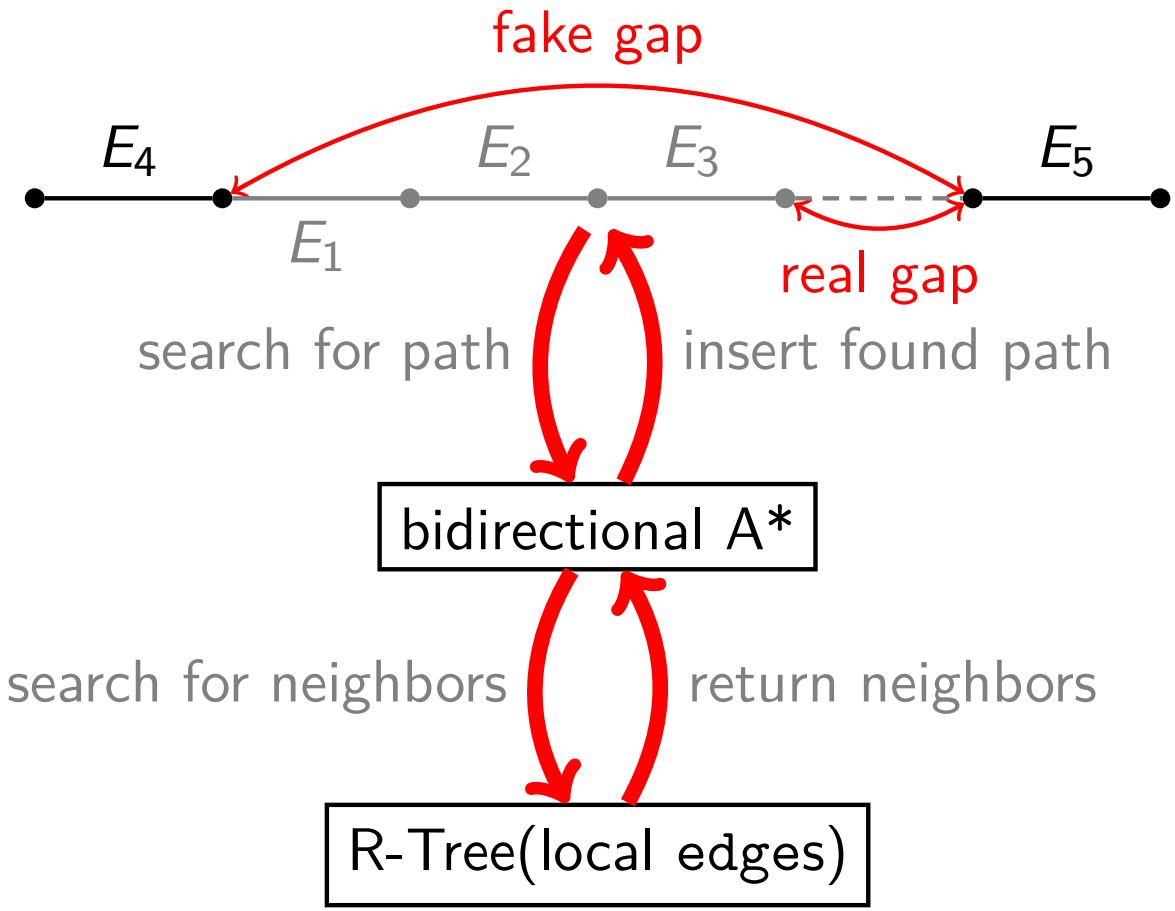
Order Repair - fixing gap between E_4 and E_5



Edge Order (after using bidirectional A*):

$$E_1 \to E_2 \to E_3 \to E_2 \to E_1 \to E_4 \to E_1 \to E_2 \to E_3 \dashrightarrow E_5$$

Order Repair - fixing gap between E_4 and E_5



Edge Order (after Order Repair):

$$E_4 \rightarrow E_1 \rightarrow E_2 \rightarrow E_3 \dashrightarrow E_5$$

Oder Repair - Procedure

Algorithm 1: Order Repair

Input: a *network* contains graphs, edges and vertexs

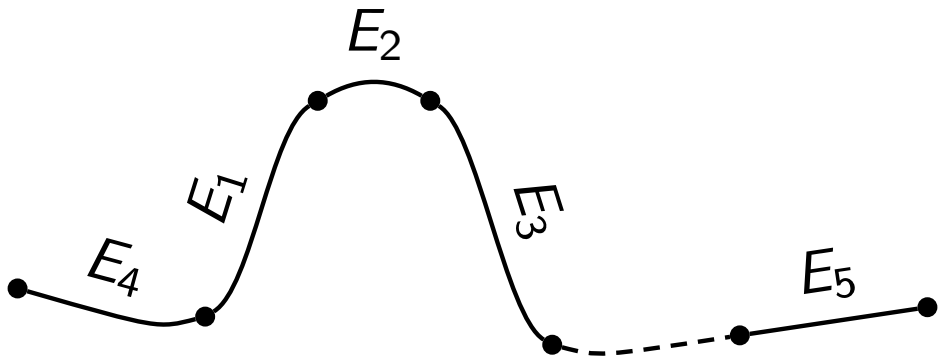
```
1 foreach graph ∈ network do
2     gap_list ← GapCheck(graph);
3     r-tree ← BuildRTreeUsingLocalEdges(graph);
4     foreach gap ∈ gap_list do
5         GapRepairUsingBiAStar(r-tree, graph, gap);
      /* find the first non-repetitive edge and delete edges front
         it
         make sure the front end-edge is in position                */
6     LocateFirstEdge(graph);
```

Oder Repair - Explanation

Questions:

1. repetitive edges?
   - key to keep the local order right
   - removable through DFS
2. theoretical basis of `LocateFirstEdge`?
   the front end-edge intersects with its neighbors at one identical conjoint point
3. why bidirectional A* algorithm?
   - fast and efficient
   - if a complete path doesn't exist, highly possible to find a partial path to transform a fake gap into a real one

Gap Repair - Introduction



—— : contained in this graph

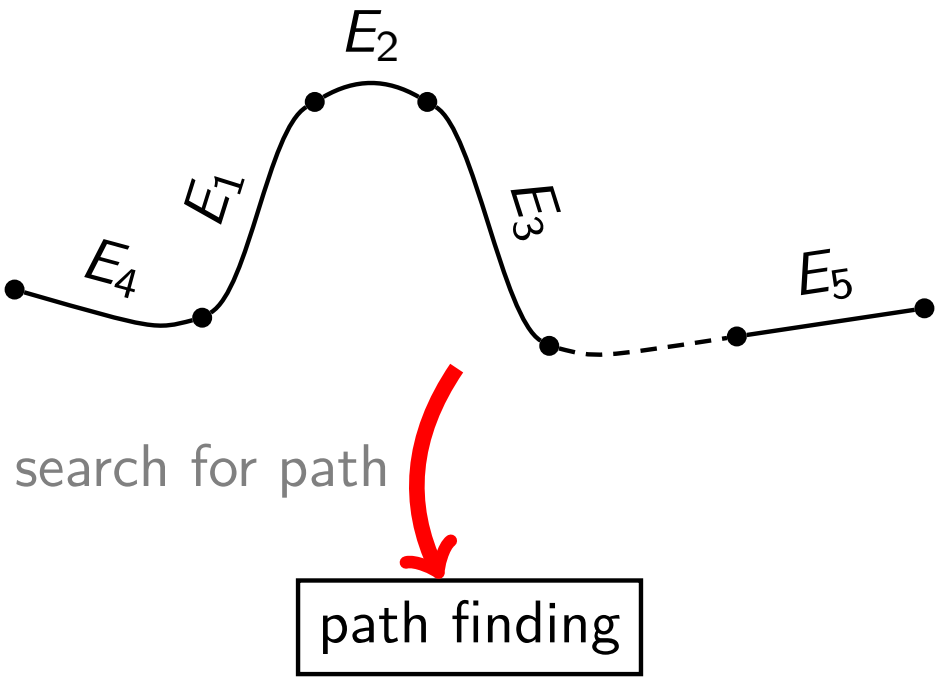
--- : not contained in this graph

Original Edge Order (containing real gaps):

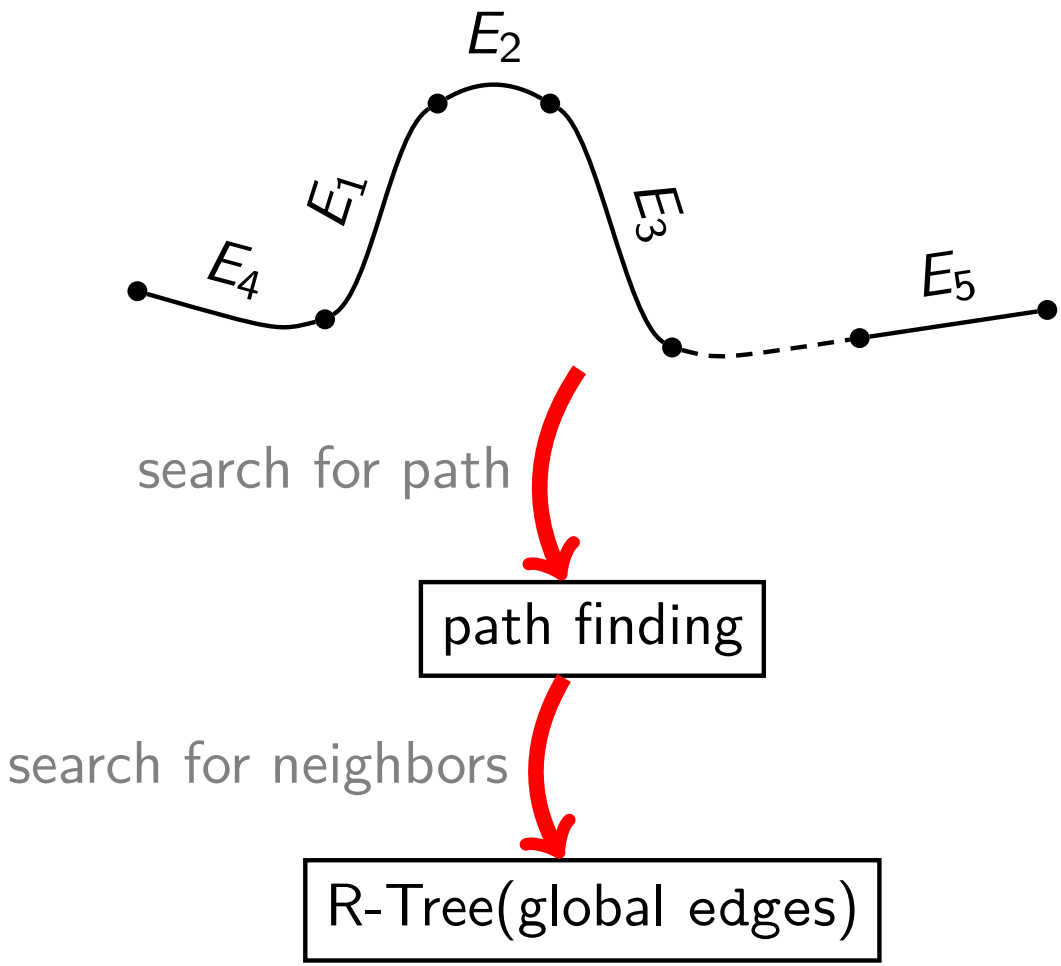
$$E_4 \rightarrow E_1 \rightarrow E_2 \rightarrow E_3 \dashrightarrow E_5$$

$\dashrightarrow$: means a gap.

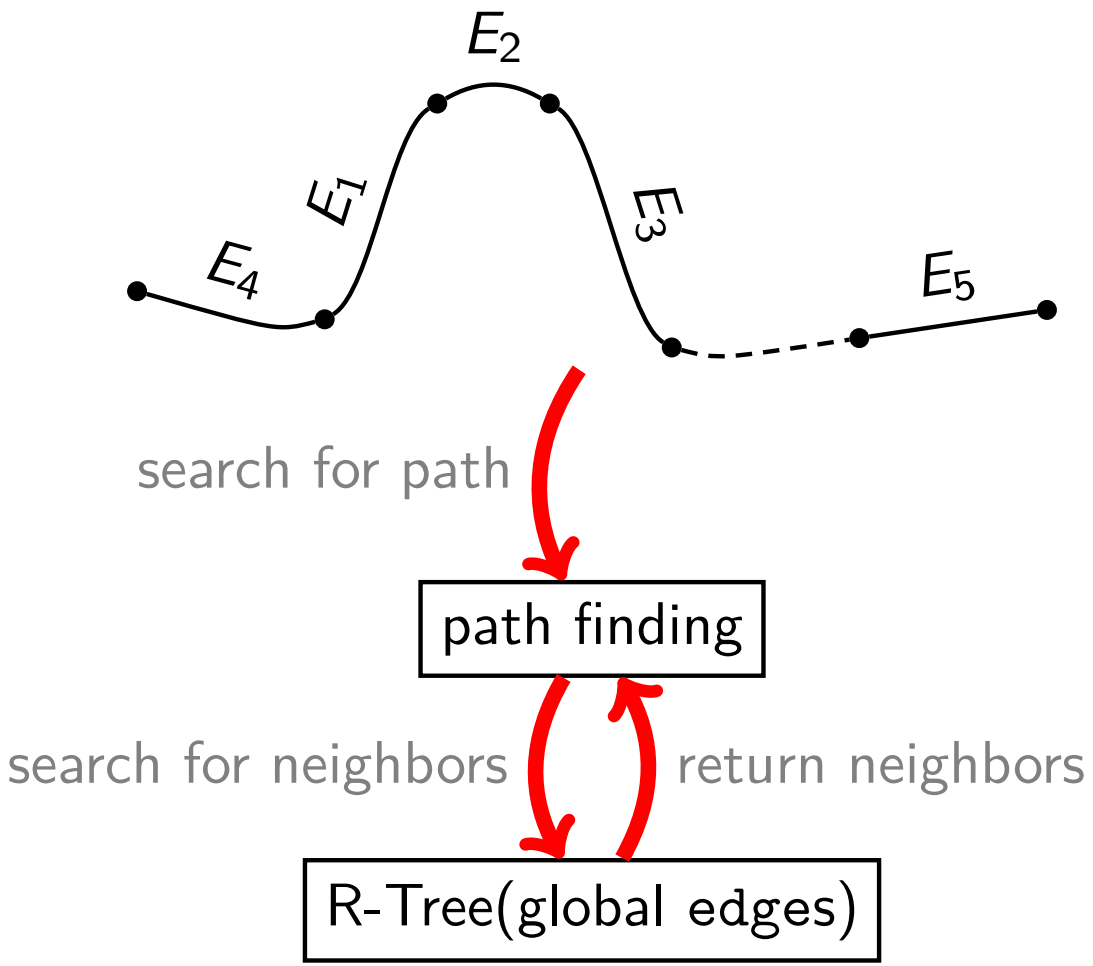
Gap Repair - Introduction



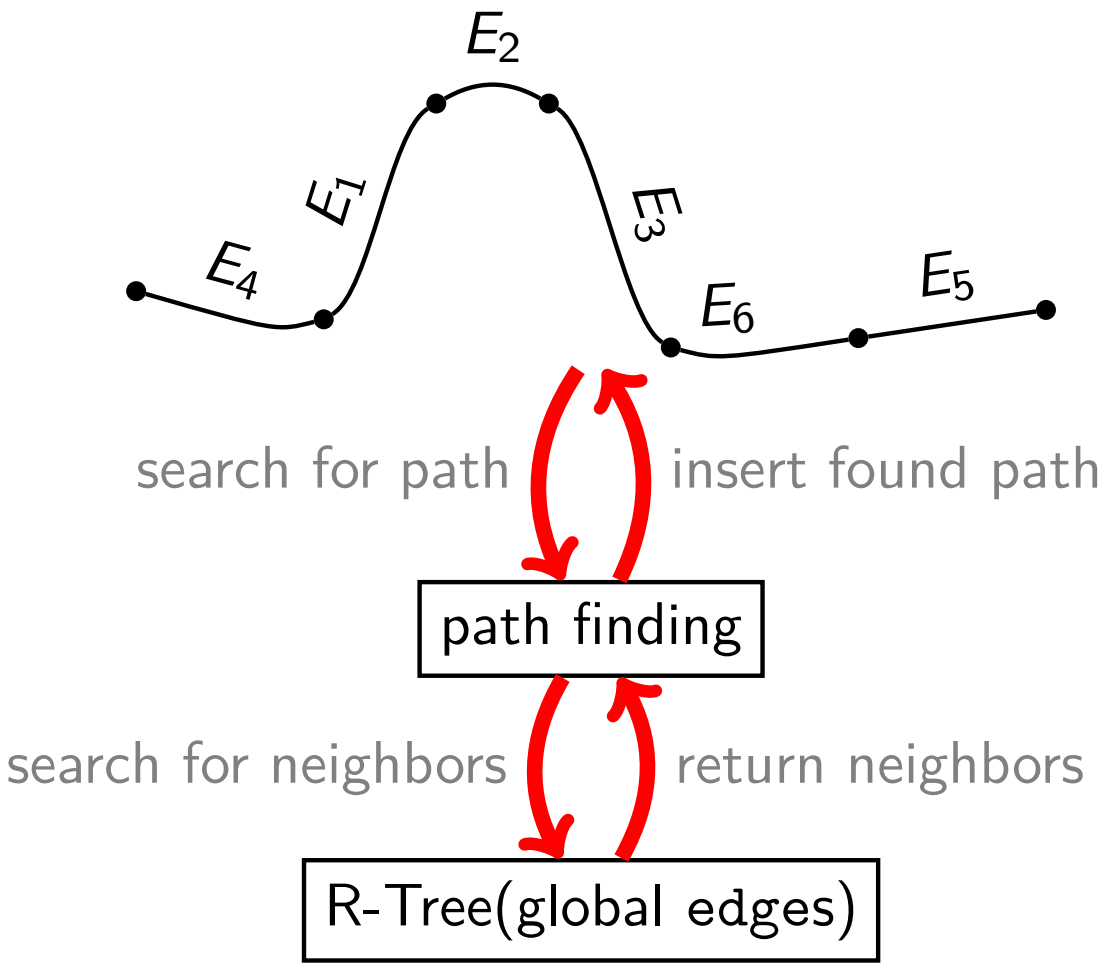
Gap Repair - Introduction



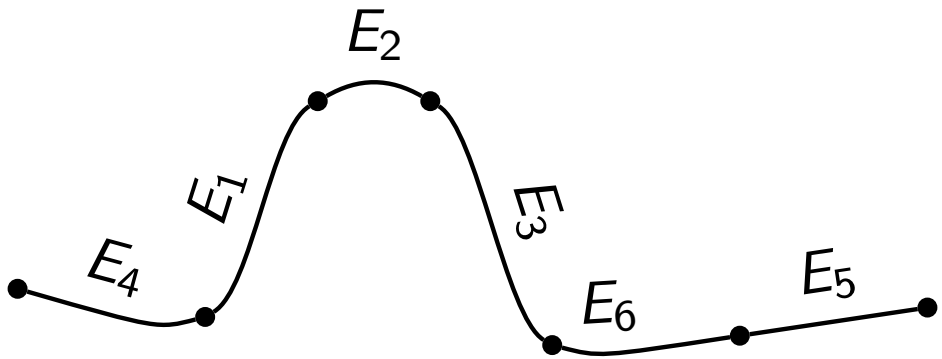
Gap Repair - Introduction



Gap Repair - Introduction



Gap Repair - Introduction



Edge Order (after Gap Repair):

$$E_4 \rightarrow E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow E_6 \rightarrow E_5$$

Gap Repair - Procedure

Algorithm 2: Gap Repair

Input: a *network* contains graphs, edges and vertexs

```
1  gap_list_dic ← empty dictionary;          // key is graph, value is a gap_list
2  foreach graph ∈ network do
3      gap_list ← GapCheck(graph);
4      gap_list_dic.Add(graph, gap_list);
5  r-tree ← BuildRTreeUsingGlobalEdges(network);  // all the edges in network
6  foreach (graph, gap_list) ∈ gap_list_dic do
7      foreach gap ∈ gap_list do
           /* GapRepair returns true/false to indicate if a gap is successfully fixed
              pathFindingAlgo could be A* Algorithm or Dijkstra's Algorithm        */
8          IsFixed ← GapRepair(r-tree, graph, gap, pathFindingAlgo);
9          if ¬IsFixed then
10             break;
```

Classification - Introduction

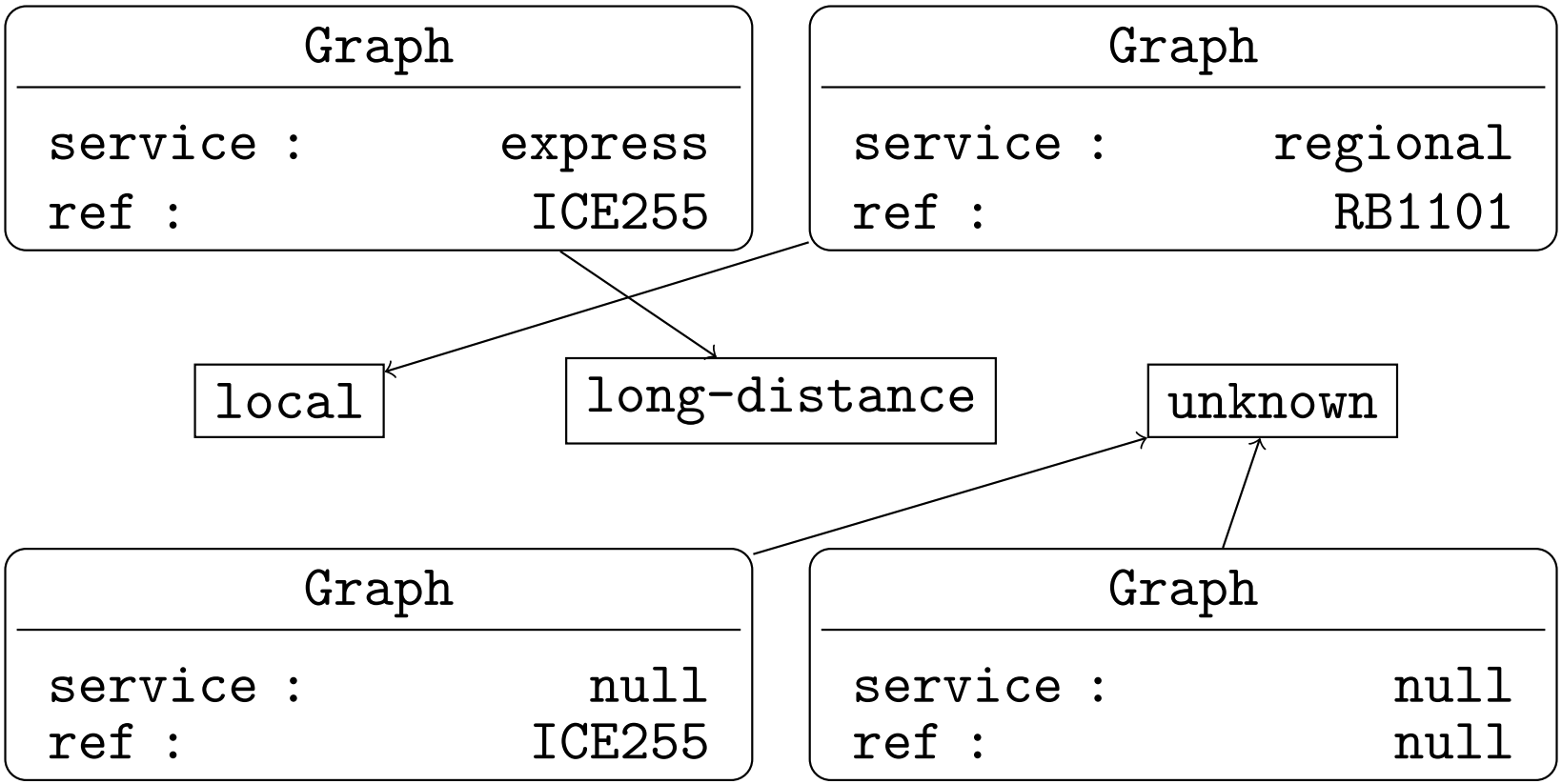
Graph	
service :	express
ref :	ICE255

Graph	
service :	regional
ref :	RB1101

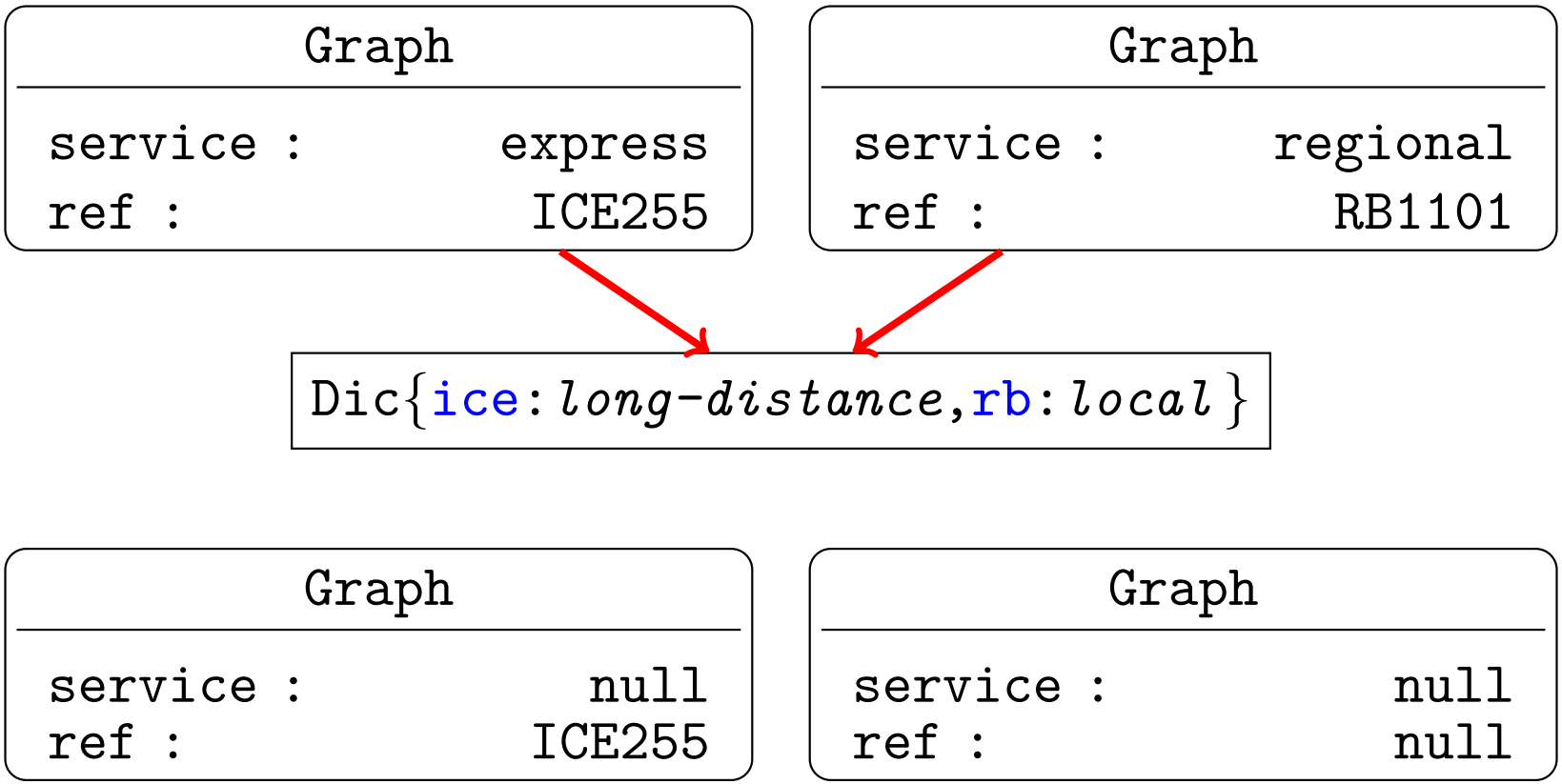
Graph	
service :	null
ref :	ICE255

Graph	
service :	null
ref :	null

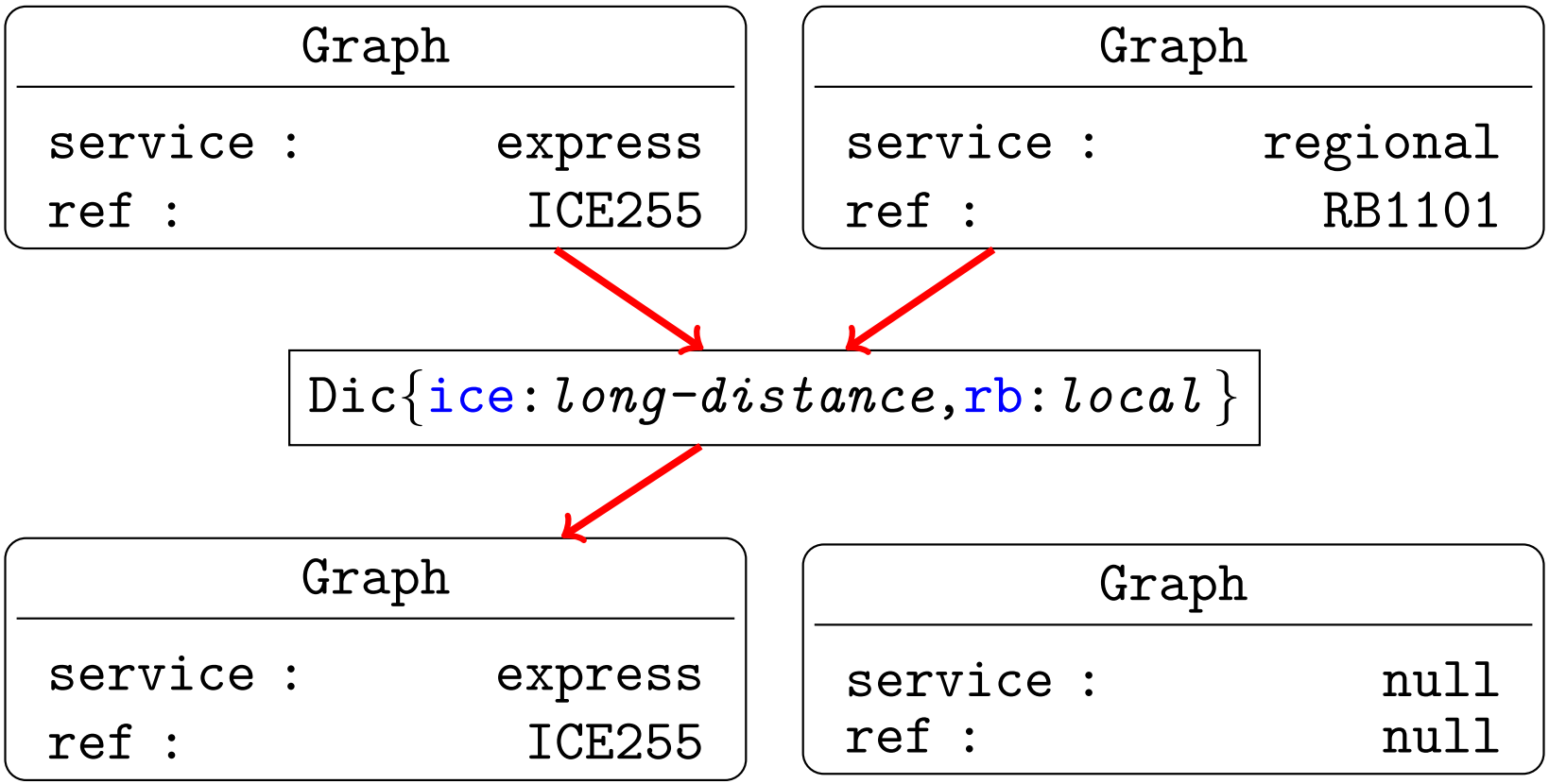
Classification - Introduction



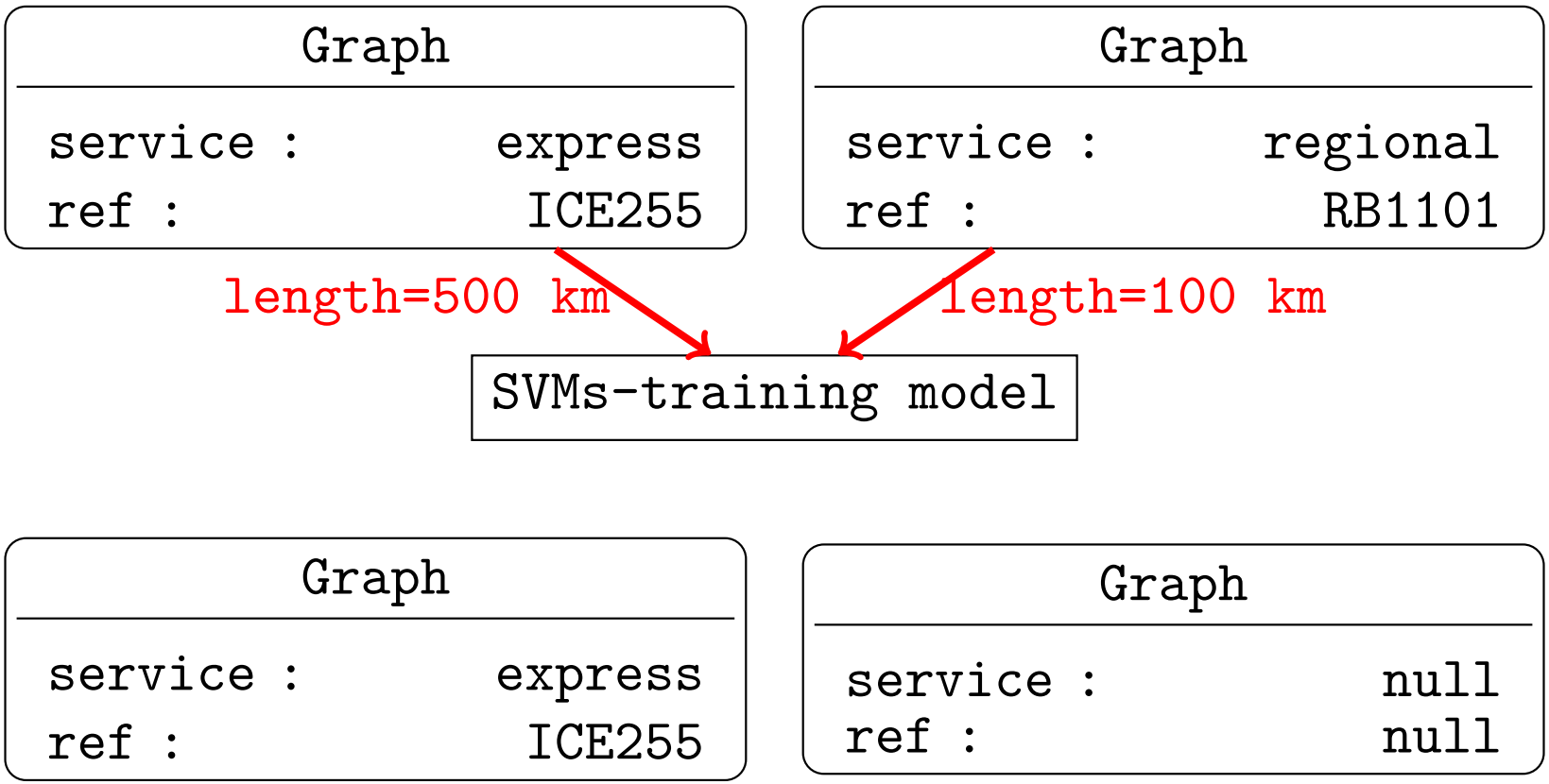
Classification - Introduction



Classification - Introduction



Classification - Introduction



Classification - Introduction

Graph	
service :	express
ref :	ICE255

Graph	
service :	regional
ref :	RB1101

SVMs-predict model

length=90 km

Graph	
service :	express
ref :	ICE255

Graph	
service :	regional
ref :	null

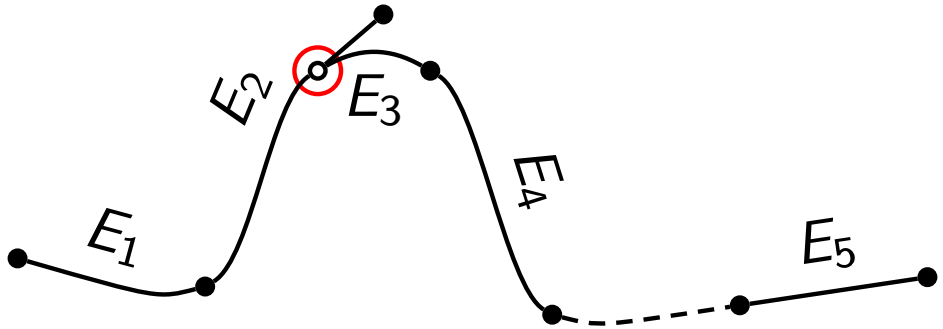
Classification - Procedure

Algorithm 3: Classification

Input: a *network* contains graphs, edges and vertexs

```
1  locals, long_distances, unknows ← empty list;
2  foreach graph ∈ network do
3      if IsLocal(graph) then
4          locals.Add(graph);
5      else if IsLongDistance(graph) then
6          long_distances.Add(graph);
7      else
8          unknows.Add(graph);
   /* get a dictionary, in which one prefix points to only one class label local or
      long-distance                                                              */
9  ref_predix_dic ← SummarizeUniqueRefPredix(locals, long_distances);
10 PredictUsingDic(ref_predix_dic, locals, long_distances, graph);
11 svms ← SVMs();
12 svms.Training(locals, long_distances);        // train SVMs with the lengths
13 PredictUsingSVMs(svms, graph);
```

Connection Repair - Introduction



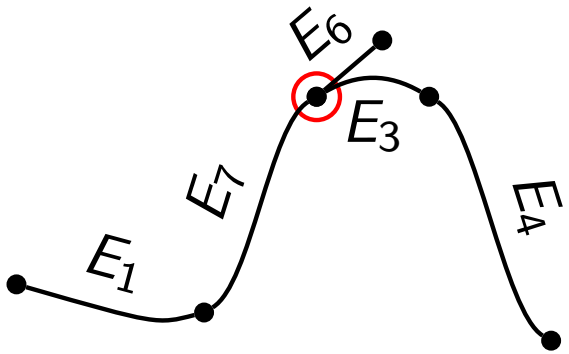
Edge Order:

$$E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow E_4 \dashrightarrow E_5$$

Vertex Order:

- E_1: $v_1 \rightarrow v_2 \rightarrow v_3$
- E_2: $v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7$
- E_3: $v_5 \rightarrow v_8 \rightarrow v_9$
- E_4: $v_{11} \rightarrow v_{10} \rightarrow v_9$
- E_5: $v_{13} \rightarrow v_{14} \rightarrow v_{15}$

Connection Repair - Introduction



Edge Order:

$$E_1 \rightarrow E_7 \rightarrow E_6 \rightarrow E_3 \dashrightarrow E_4$$

Vertex Order:

- E_1: $v_1 \rightarrow v_2 \rightarrow v_3$
- E_7: $v_3 \rightarrow v_4 \rightarrow v_5$
- E_3: $v_5 \rightarrow v_8 \rightarrow v_9$
- E_4: $v_9 \rightarrow v_{10} \rightarrow v_{11}$
- E_6: $v_5 \rightarrow v_6 \rightarrow v_7$

Connection Repair - Procedure

Algorithm 4: Connection Repair

Input: a *network* contains graphs, edges and vertexs

```
1  foreach graph ∈ network do
2      foreach edge_front, edge_back ∈ graph do
           /* edge_front and edge_back are index-adjacent in graph */
3          if IsConjoint(edge_front, edge_back) then
4              if ¬IsHeadTailType(edge_front, edge_back) then
                   /* make sure they are conjoint in head-tail type.  If
                      necessary, separate one of them to achieve it */
5                  SepatateEdge(edge_front, graph) or
                   SepatateEdge(edge_back, graph);
6          else
7              foreach edge ∈ graph ∧ edge behind edge_back do
8                  graph.DeleteEdge(edge);
```

Topological Sort - Introduction

Problems:

- repetitive edges exist in `graph`
- edges in `graph` are poor organized

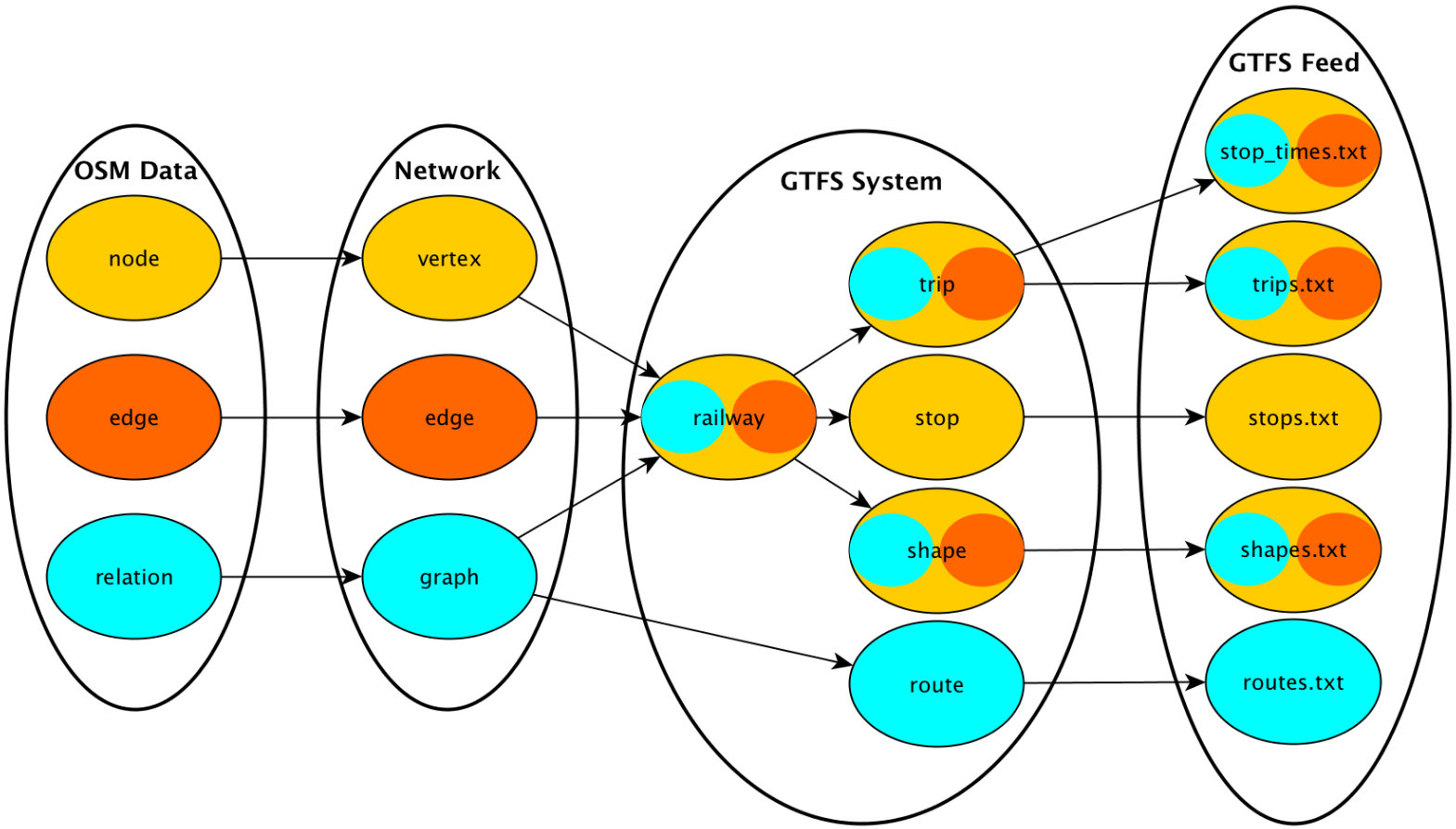
Aim: each `graph` has a topologically sorted edge-list.

Algorithm 5: Topological Sort

Input: a *network* contains `graphs`, `edges` and `vertexs`

```
1 foreach graph ∈ network do
2     adjacency_list ← CreateAdjacencyLists(graph);
3     DFS(adjacency_list, graph);
```

Generating GTFS Feed - Introduction



Generating GTFS Feed - Procedure

Algorithm 6: Generating GTFS Feed

Input: a *network* contains graphs, edges and vertexs
Output: a GTFS feed

```
1 GTFSSystem ← GTFSSystem();
2 r-tree ← BuildRTreeUsingGlobalStations(network);
3 foreach graph ∈ network do
4     route ←Route(graph);                    // store the attributes of graph
5     GTFSSystem.Add(route);
6     railway_list ← SeparateIntoRailways(graph);     // create several
      railways, each of them could store the information of a branch of graph
7     foreach railway ∈ railway_list do
          /* separate the information in a railway into stops, trips and shape
             */
8         ProcessRailway(GTFSSystem, railway, r-tree);
9 OutputGTFSFeed(GTFSSystem);     // output a GTFS feed using a CSV writer
```

Table of Contents

Evaluation - Part I

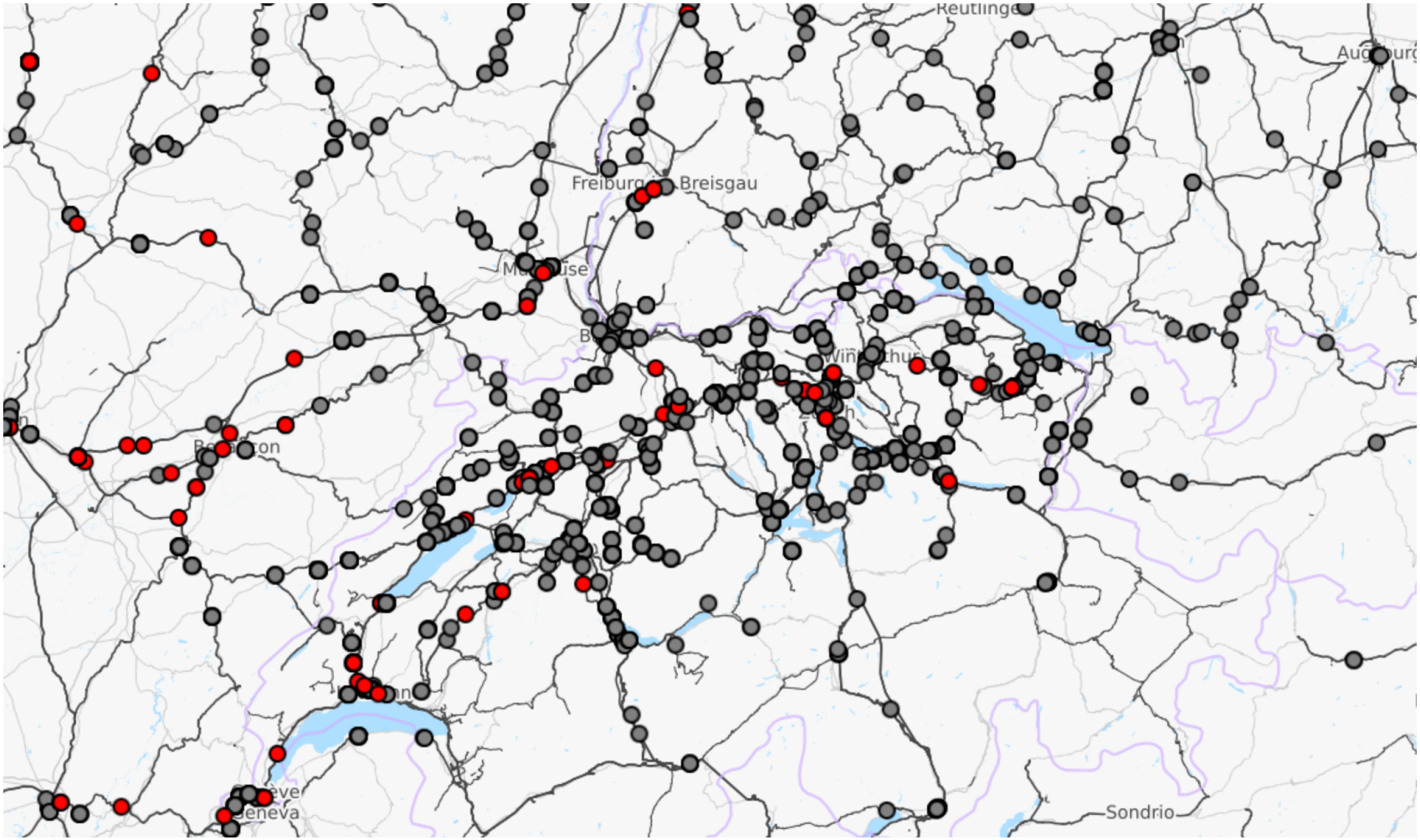
		GERMANY				EUROPE	PLANET
		A* & O.R.	A*	Dijkstra's & O.R.	Dijkstra's	A* & O.R.	A* & O.R.
Graph Number		2022				7959	12936
Edge Number		127244				439244	688197
Vertex Number		788464				4227864	8290301
Runtime (ms)	Data Extraction	1262728	1186926	1246597	1213944	112001	217700
	Order Repair	1967498	0	1951216	0	39941981	73918960
	Gap Repair	683592	3557606	558782	5051321	5419185	10775399
	Classification	1222	1212	1553	1245	5589	9320
	Connection Repair	1709	1615	2347	1477	19268	28954
	Topological Sort	330	309	368	286	2295	3479
	Generating GTFS Feed	14907	17812	17359	17992	81344	109627
Gap Number	Removed by OR	9315	0	9315	0	138727	181349
	Removed by GR	162	1955	163	2129	669	814
	Total	23243				180886	235644
Gap-repaired Rate	Order Repair	40,0 %	0,0 %	40,0 %	0,0 %	76,7 %	77,0 %
	Gap Repair	0,7 %	8,4 %	0,7 %	9,2 %	0,3 %	0,3 %
	Total	40,7 %	8,4 %	40,7 %	9,2 %	77 %	77,3 %

* O.R. is ab. for Order Repair.

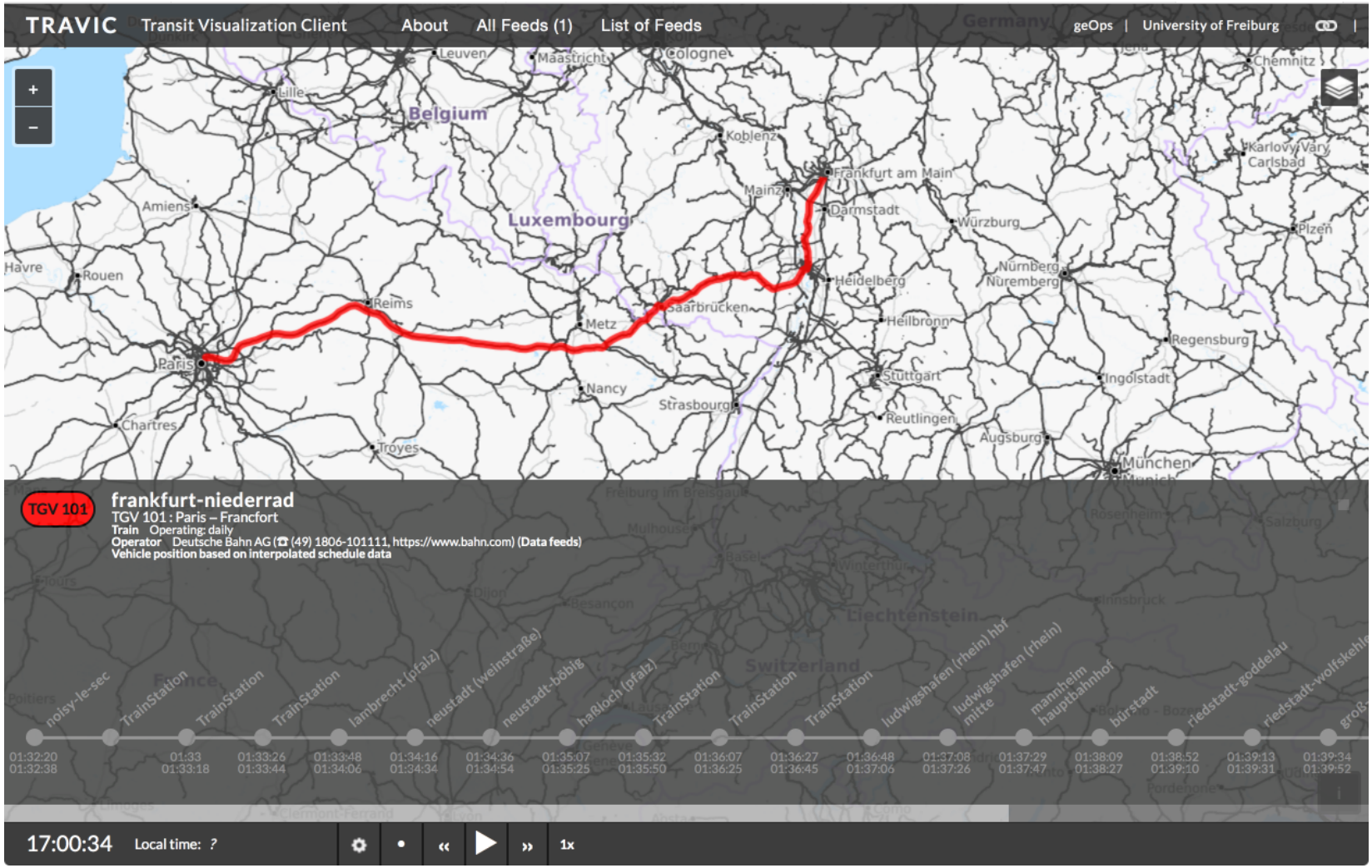
Conclusions:

- using A* $\iff$ using Dijkstra
- Order Repair is efficient in removing gaps.

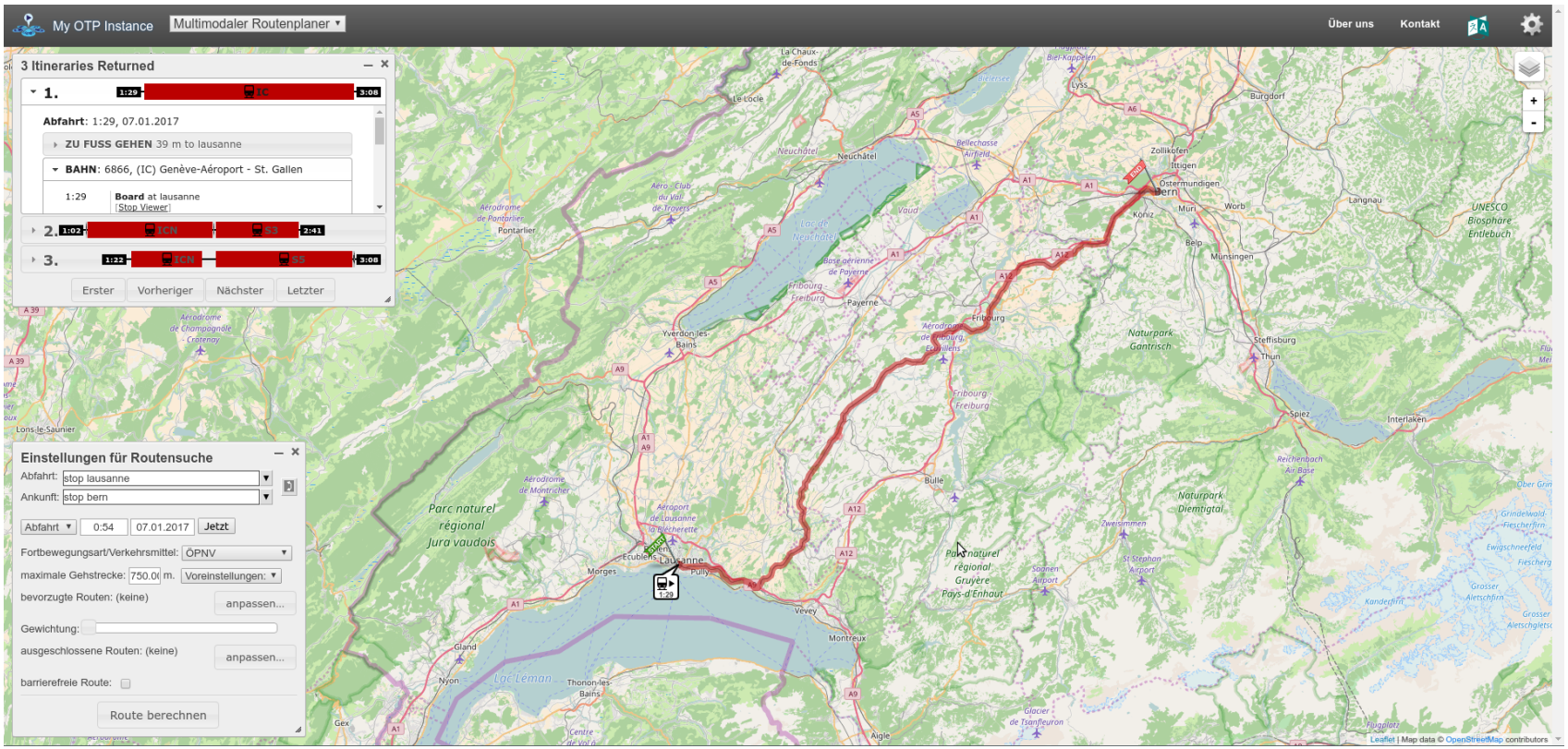
Evaluation - TRAVIC I



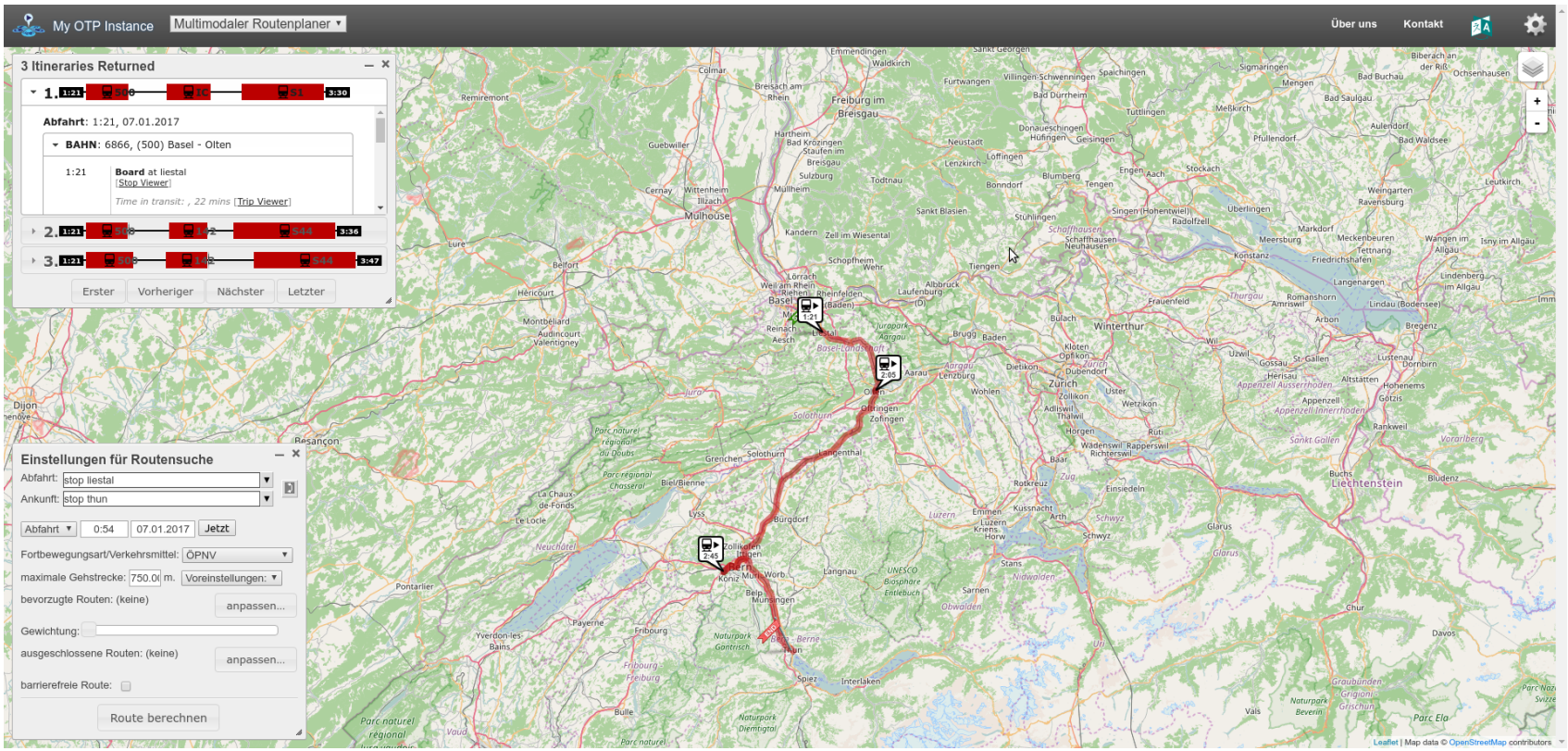
Evaluation - TRAVIC II



Evaluation - OTP I



Evaluation - OTP II



Evaluation - Part II

Conclusions:

- result in TRAVIC is very good.
- result in OTP is ok. For some routes is fine.

Analysis:

- We take the condition into account, that some stops belong to a big station. So the GTFS feed contains this information.
- OTP doesn't recognize this information in the GTFS feed. It leads to a not good result.

Table of Contents

Further Research

1. Extending network by collecting all the ways near to at least one of the railways
2. Improving the efficiency of classification by precomputing training data
3. Identifying neighbors with regard to the existence of a real gap

Thank you for your attention!

Question?