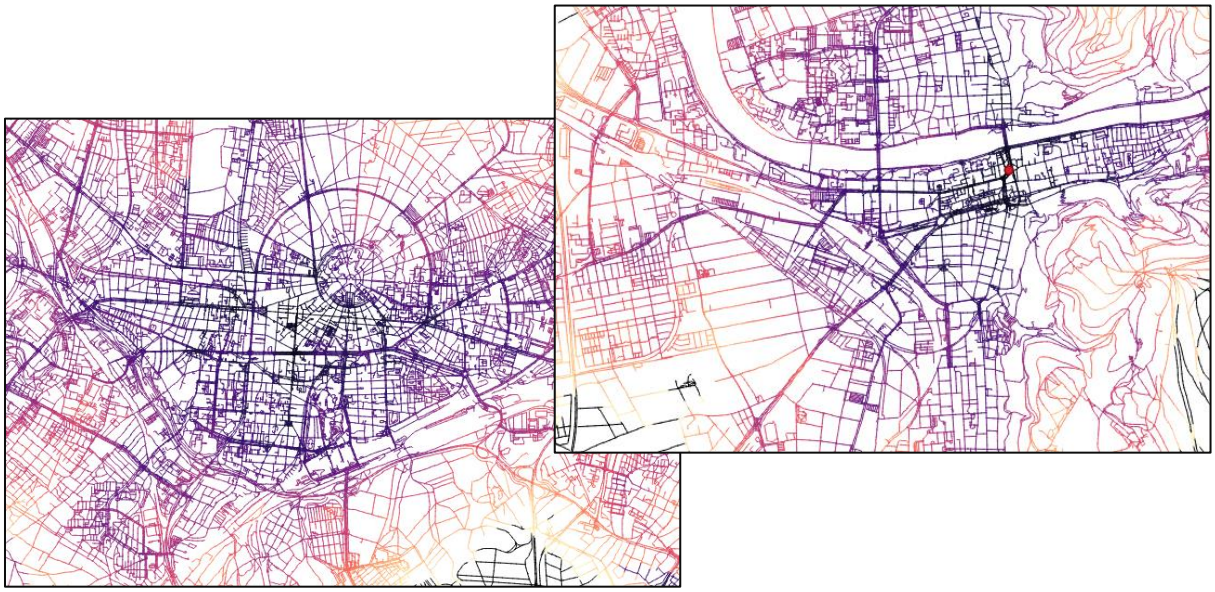
Ruprecht-Karls University Heidelberg

Faculty of Chemistry and Earth Sciences

Institute of Geography

Winter Term 2018/2019



Connectivity analysis for cities including public transportation

Czizewski Philippé

MtkNr. : 3385951

czizewski@stud.uni-heidelberg.de

Stier Jochen

MtkNr. : 3381546

j.stier@stud.uni-heidelberg.de

lecture: GIS analyses with free and open source software

lecturers: Julian Bruns, Christina Ludwig

15.04.2019

Table of contents

List of dependencies

GRASS 7.4.0

QGIS 3.4.1

1. Introduction

Public transportation and low departure times are crucial topics for the development process of modern cities as populated areas tend to grow even larger in the future, while many local inhabitants are still dependent on public infrastructure with short transportation times to reach their workspace and public institutions on regular basis. Besides most urban areas with a high car dependency face challenges such as increased travel periods due to congestion as well as noise and air pollution. With the ban of diesel vehicles in large German cities and the rising awareness in climate protection, the importance of public transportation in major cities increases even more.

Therefore our analysis focuses on the accessibility of civic centers in selected cities including the local public transportation. The script analyzes the infrastructure network in a defined study area on the example of Heidelberg and Karlsruhe and gives information about how fast the user can reach the civic center from the remaining city area by public transportation and foot.

In order to introduce our analysis in this paper we will first present our selected study area as well as the data sets and data sources used to perform the script.

Following we will walk through the script in detail and describe the performed steps and their effect as part of the workflow.

After displaying the solved problems which occurred during the processing of the script, we will end the paper with an evaluation which will show possible improvements and the limitations of our analysis.

2. Study area and data sets

As primary study area we chose Heidelberg and Karlsruhe, on which we also tested the script of our analysis. We decided to focus on these two cities because they are mapped on OpenStreetMap with great detail and accuracy. Furthermore both cities possess a well developed public transportation network with various bus and tram lines. The size of the cities was ideal for our analysis since they lack on substantial subway or train lines, which are currently not included in the script.

We obtained the necessary OpenStreetMap data sets for our analysis using the Overpass turbo API. By generating multiple queries with different tags, we gathered data about the city's respective street network as well as data sets for public transportation lines and stops (e.g. fig. 1).

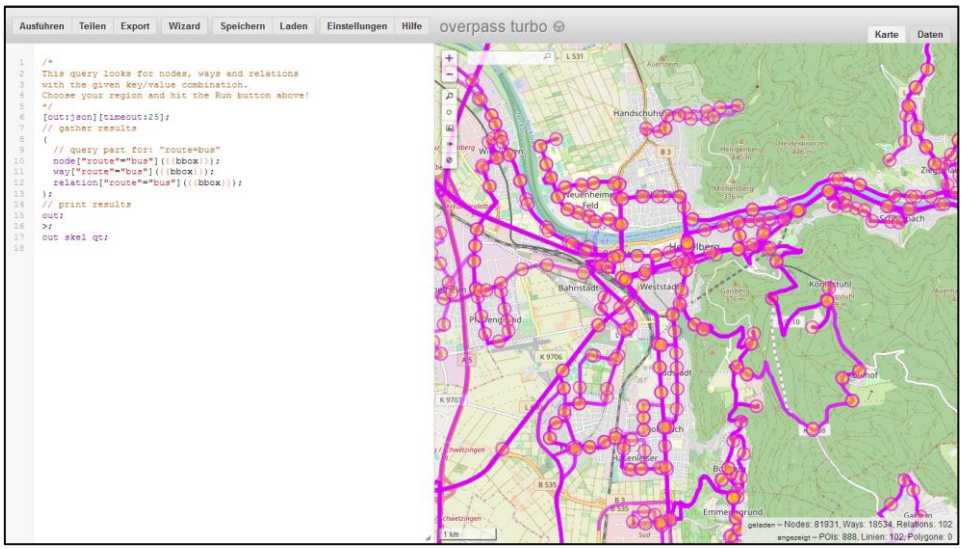


Figure 1: Overpass turbo query for buses in Heidelberg

Since the workflow requires a certain civic center to perform, we chose the Bismarckplatz for Heidelberg and the Europaplatz for Karlsruhe as central point for their respective city. These town squares are particularly well suited for the analysis because they are connected to the majority of bus and tram routes inside the study areas, as well as being highly frequented on a daily basis. Unfortunately these civic centers weren't mapped as nodes in OpenStreetMap, for which reason we manually digitized both of them as points by using the toolbox of QGIS. The resulting shapefiles were later implemented in GRASS GIS as part of the analysis.

To keep account of the unequal transportation speed of the three primary transportation methods (bus, tram and by foot) included in the script, we manually added the average speed times in an own column. For a mixed route by bus and tram we simply calculated the average speed for both transportation methods.

3. Analysis workflow

Now we will present the workflow of our analysis by explaining the code of the script used to perform the analysis:

Before the data preparation and import can be started, the necessary GRASS tools and library for importing data into python had to be prepared. To prevent data corruption by old vector data from previous iterations, the data base elements from the user's current mapset get removed by the 'g.remove' command (fig. 2).

```
3  import grass.script as grass                #auto
4  import os                                   #for importing data
5
6  def main():
7      grass.run_command('g.region', flags='p') #auto^M
8      grass.run_command('g.remove', flags='f' , type='vector', pattern='*')
```

Figure 2: Analysis preparation

I. Importing and preparing data

At first the initial data had to be prepared and imported into Python to perform the analysis. For that reason three different base paths got defined, which were later used to load and store data. The cache is necessary for the assignment of unique IDs, the data stored here is of no further interest.

Bus route data sets are added as lines and bus stops as points, which is also done with tram routes and stops. The street network was integrated as lines as well, while the civic centers got assigned as points (fig. 3).

```
#sets basepath
input_path = "C:/Users/jocho/Desktop/opengis/project/data/in"   #for input data
inter_path = "C:/Users/jocho/Desktop/opengis/project/data/inter"#cache
out_path = "C:/Users/jocho/Desktop/opengis/project/data/out"    #output data

#bus -> route as lines, stops as points
path_bus_stop=os.path.join(input_path, 'bus', 'bus_stop.shp')
grass.run_command('v.in.ogr', input=path_bus_stop, output='busstop', overwrite=True, cnames=True)

path_bus_route=os.path.join(input_path, 'bus', 'bus_route.shp')
grass.run_command('v.in.ogr', input=path_bus_route, output='busroute', overwrite=True, cnames=True)

#tram -> route as lines, stops as points
path_tram_stop=os.path.join(input_path, 'tram', 'tram_stops.shp')
grass.run_command('v.in.ogr', input=path_tram_stop, output='tramstop', overwrite=True, cnames=True)

path_tram_route=os.path.join(input_path, 'tram', 'tram_route.shp')
grass.run_command('v.in.ogr', input=path_tram_route, output='tramroute', overwrite=True, cnames=True)

#streets as lines
path_streets=os.path.join(input_path, 'streets', 'streets.shp')
grass.run_command('v.in.ogr', input=path_streets, output='streets', overwrite=True, cnames=True)

#central point
path_central_point=os.path.join(input_path, 'central_point', 'central_point.shp')
grass.run_command('v.in.ogr', input=path_central_point, output='central_point', overwrite=True, cnames=True)
```

Figure 3: Base paths & data import

In order to enable a switch between tram and bus routes with a mixed route of both transportation methods, both public transportation lines were patched together. This also expands the public transportation net by areas, buses and trams can't reach themselves in the same time compared with a mixed route. The same was done with the civic centers and stops, which is a prerequisite for the 'v.net.iso' command used later in the analysis (fig. 4).

```

#patching bus and tram routes together:
    grass.run_command('v.patch', input=['tramroute','busroute'], output='tbroute', overwrite=True)
    grass.run_command('v.patch', input=['busstop','tramstop'], output='tbstop', overwrite=True)


#connecting central point to stops
    grass.run_command('v.patch', input=['central_point','tramstop'], output='c_tramstop', overwrite=True)
    grass.run_command('v.patch', input=['central_point','busstop'], output='c_busstop', overwrite=True)
    grass.run_command('v.patch', input=['central_point','tbstop'], output='c_tbstop', overwrite=True)
    grass.run_command('v.db.addtable', map='tbstop',overwrite=True)
```

Figure 4: combining lines and points

‘v.net.iso’ uses a range of points as centers from which the distance across the network is measured. Trough adding the civic center to the stops, the operator “connect” of ‘v-iso’ connects all public transportation stops and the center point, which in turn enables ‘v.distance’ to safely connect the correct time stamps to each stop. The time stamps get stored in an own table, which is created alongside the connection of the civic center and stops.

After that all points for the bus and tram stops get exported and imported again, to assign an individual ID to each point (fig. 5). Since the public transportation stop layers are patched from two different layers, the first ID of both layers would overlap which could cause an error with the ‘v.net.iso’ command.

```
#exporting/importing to write features with cat 1 into seperate features -> otherwise 2 have the same id
    grass.run_command('v.out.ogr',input='c_tramstop',output=inter_path, format='ESRI_Shapefile',overwrite=True)
    path_c_tramstop=os.path.join(inter_path, 'c_tramstop.shp')
    grass.run_command('v.in.ogr', input=path_c_tramstop, output='c_tramstop', overwrite=True, cnames=True)
    grass.run_command('v.out.ogr',input='c_busstop',output=inter_path, format='ESRI_Shapefile',overwrite=True)
    path_c_busstop=os.path.join(inter_path, 'c_busstop.shp')
    grass.run_command('v.in.ogr', input=path_c_busstop, output='c_busstop', overwrite=True, cnames=True)
    grass.run_command('v.out.ogr',input='c_tbstop',output=inter_path, format='ESRI_Shapefile',overwrite=True)
    path_c_tbstop=os.path.join(inter_path, 'c_tbstop.shp')
    grass.run_command('v.in.ogr', input=path_c_tbstop, output='c_tbstop', overwrite=True, cnames=True)


#cleaning vectors -> removes unconnected lines (rmdangle), duplicates(rmdupl), parallel lines within a certain distance(snap) and connects lines at every intersection
(break)
    grass.run_command('v.clean', input='tramroute', output='tramroute2', tool=['snap','break','rmdupl','rmdangle'],overwrite=True, threshold=[30,0,0,30])
    grass.run_command('v.clean', input='busroute', output='busroute2', tool=['snap','break','rmdupl','rmdangle'],overwrite=True, threshold=[30,0,0,30])
    grass.run_command('v.clean', input='tbroute', output='tbroute2', tool=['snap','break','rmdupl','rmdangle'],overwrite=True, threshold=[30,0,0,30])
    grass.run_command('v.clean', input='streets', output='streets2', tool=['break','rmdupl','rmdangle'],overwrite=True, threshold=[0,0,30])
```

Figure 5: Assigning individual IDs & cleaning vectors

Additionally the street and public transportation network gets cleaned by removing unconnected paths, duplicates and parallel lines (snaps) as well as connecting lines at every intersection. Connecting the separate lines by using the ‘break’ tool is crucial for the analysis, otherwise only lines that go through the central point would be considered drivable by our script. This step wouldn’t be necessary if we had another solution to take account for the change in lines.

II. Network 1

Like this the initial data is prepared and ready to be used by constructing the first network within the analysis. The first network is primary used to calculate the time necessary to reach each stop from the civic center.

The command ‘v.net’ creates a net layer using the points and lines of the different public transportation layers for bus and tram despite the difference in the data’s dimension (fig. 6). The operator ‘connect’ links unconnected points with their closest point in a direct line to create a gapless public transportation net.

```
#conntecting central point to lines
    grass.run_command('v.net', input='busroute2', points='c_busstop', output='busnet', operation='connect', threshold = 20, overwrite=True, alayer=1,nlayer=2)
    grass.run_command('v.net', input='tramroute2', points='c_tramstop', output='tramnet', operation='connect', threshold = 20, overwrite=True,alayer=1,nlayer=2)
    grass.run_command('v.net', input='tbroute2', points='c_tbstop', output='tbnet', operation='connect', threshold = 20, overwrite=True,alayer=1,nlayer=2)
```

Figure 6: Connecting central point to lines

Following that we further prepare our first network analysis. Our command ‘v.net.iso’ requires ‘costs’ as an input which is measured in meters. If you would set a range between 3 – 5 the command would simply split the net into three categories: lines which are under three meters away from the civic center (cat 1), lines between three and five meters (cat 2) as well as lines that are even further away (cat 3).

```
78  #Preparing lists that are classified by traveled distance in meters with different methods.^M
79      costs_tram=[]   #trams travel at 295m/min-> input for costs^M
80      costs_bus=[]    #buses travel at 233m/min^M
81      costs_foot=[]   #traveling by foot comes in at 67m/min^M
82      costs_tb=[]^M
83      temp_tram=295^M
84      speed_tram=295^M
85      temp_bus=233 ^M
86      speed_bus=233^M
87      temp_foot=67^M
88      speed_foot=67^M
89      temp_tb=264^M
90      speed_tb=264^M
91      ^M
92      for i in range(1,99):       #-> 99 categories and +1 in category means +x traveled distance
93          costs_tram.append(temp_tram)^M
94          costs_bus.append(temp_bus)^M
95          costs_foot.append(temp_foot)^M
96          costs_tb.append(temp_tb)^M
97          temp_tram=temp_tram+speed_tram^M
98          temp_bus=temp_bus+speed_bus^M
99          temp_foot=temp_foot+speed_foot^M
100         temp_tb=temp_tb+speed_tb^M
```

Figure 7: Preparing costs for 'v.net.iso' with average speed times

The code above creates a list which classifies the lines according to the average speed of the public transportation methods (fig. 7). For example: A tram in Heidelberg has an average speed of 295 meters per minute. By that it takes up to one minute for a tram to reach a stop which is 0 to 295 meters away, to travel another 295m it takes an additional minute. If we now input a list which contains 295^{x} meters (**295**, **590**(*295+295*), **885**(*295+295+295*)….) as costs, it splits our net into x different categories (cat 1,2,3…) who match the time necessary to cross the respective distance. If a point on a line has the forth category, it would take four minutes to reach that point from the civic center – for example.

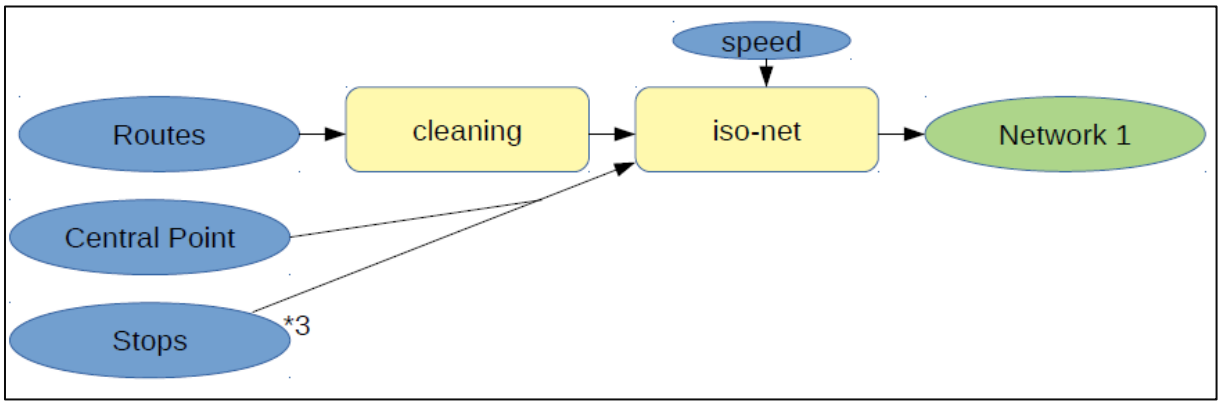


Figure 8: Workflow Network 1

After this we are able to perform our analyses for the first network between the civic center and the public transportation network (fig. 8). The command ‘v.net.iso’ splits the net into bands between cost isolines (fig.9). ‘center_cat 1’ is our initial civic center, the costs were calculated in the previous step.

```
96  #network analysis of from central point along the lines^M
97      grass.run_command('v.net.iso', input='tramnet', output='iso_tram',center_cats=[1], costs=costs_tram, overwrite=True, nlayer=2)^M
98      grass.run_command('v.net.iso', input='busnet', output='iso_bus',center_cats=[1], costs=costs_bus, overwrite=True, nlayer=2)^M
99      grass.run_command('v.net.iso', input='tbnet', output='iso_tb',center_cats=[1], costs=costs_tb, overwrite=True, nlayer=2)^M
```

Figure 9: First network analysis 'v.net.iso'

III. Extracting categories from Network 1

Since ‘v.net.iso’ doesn’t automatically create a table filled with the created category numbers, we have to perform this step manually. The table is used to assign the category numbers to the closest stops at hand (fig.10).

After this we add columns for the stops, which will be filled with the categories of the closest public transportation line. This is necessary in preparation for the command ‘v.distance’ which connects the public transportation net with the stations while keeping the category numbers.

```
#creating table with category numbers
    grass.run_command('v.db.addtable', map='iso_tram',overwrite=True)
    grass.run_command('v.db.addtable', map='iso_bus',overwrite=True)
    grass.run_command('v.db.addtable', map='iso_tb',overwrite=True)

#adding column
    grass.run_command('v.db.addcolumn', map='tramstop', columns='first_tram_distance integer',overwrite=True)
    grass.run_command('v.db.addcolumn', map='busstop', columns='first_bus_distance integer',overwrite=True)
    grass.run_command('v.db.addcolumn', map='tbstop', columns='first_tb_distance integer',overwrite=True)

#connecting network with stations to keep catnum
    grass.run_command('v.distance',from_='tramstop', to='iso_tram', upload='cat',column='first_tram_distance', overwrite=True)
    grass.run_command('v.distance',from_='busstop', to='iso_bus', upload='cat',column='first_bus_distance', overwrite=True)
    grass.run_command('v.distance',from_='tbstop', to='iso_tb', upload='cat',column='first_tb_distance', overwrite=True)
```

Figure 10: Add Category number table

IV. Network 2

The second network is used to calculate the necessary time to travel from any point in the study area to the closest public transportation station by foot.

Again the command ‘v.net’ is used to create a network, this time between the public transportation stops and street network (fig.11).

```
#connecting tramstops to streets
    grass.run_command('v.net', input='streets2', points='tramstop', output='streets_tram', operation='connect', threshold = 20, overwrite=True, alayer=1,nlayer=2)
    grass.run_command('v.net', input='streets2', points='busstop', output='streets_bus', operation='connect', threshold = 20, overwrite=True, alayer=1,nlayer=2)
    grass.run_command('v.net', input='streets2', points='tbstop', output='streets_tb', operation='connect', threshold = 20, overwrite=True, alayer=1,nlayer=2)
```

Figure 11: Connecting tram stops to streets, Network 2

Similar to before a list is created which is fed to ‘v.net.iso’ as ‘center_cats’ (fig 12). The list contains ints from 1 to 999 – since all points in the second network are stops, every point in the network is considered as center.

Like this the resulting network category, which is calculated by the second network analysis in the following step, now correctly depicts the necessary time from any position in the city towards the closest bus or tram stop (fig.13).

```
#Preparation for iso_streets_*
    all_centers=[]                #this part creates a list x of ints from 1-999 as an input for center_cats -> all stops are center_cats
    for i in range(1,999):
        all_centers.append(i)

#iso from stops into streets
    grass.run_command('v.net.iso', input='streets_tram', output='iso_streets_tram',center_cats=all_centers, costs=costs_foot, overwrite=True, nlayer=2)
    grass.run_command('v.net.iso', input='streets_bus', output='iso_streets_bus',center_cats=all_centers, costs=costs_foot, overwrite=True, nlayer=2)
    grass.run_command('v.net.iso', input='streets_tb', output='iso_streets_tb',center_cats=all_centers, costs=costs_foot, overwrite=True, nlayer=2)
```

Figure 12: Preparation and second network analysis

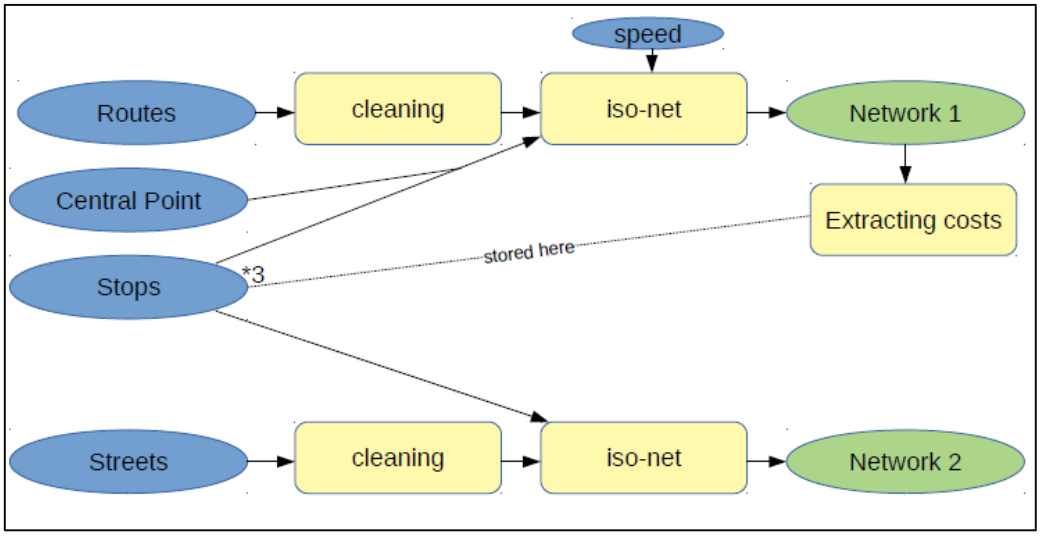


Figure 13: Workflow Network 2

V. Calculating the shortest time

After that the net layers of the second network analysis get exported and imported to convert them into vector features and split unique features which possess the same category ID (fig.14).

```
#exporting and then importing to convert net into vector feature and to split features with the same cat
    #tram
    grass.run_command('v.out.ogr',input='iso_streets_tram',output=inter_path, format='ESRI_Shapefile',overwrite=True)
    path_streets_tram_cat=os.path.join(inter_path, 'iso_streets_tram.shp')
    grass.run_command('v.in.ogr', input=path_streets_tram_cat, output='streets_tram_cat2', overwrite=True, cnames=True)
    #bus
    grass.run_command('v.out.ogr',input='iso_streets_bus',output=inter_path, format='ESRI_Shapefile',overwrite=True)
    path_streets_bus_cat=os.path.join(inter_path, 'iso_streets_bus.shp')
    grass.run_command('v.in.ogr', input=path_streets_bus_cat, output='streets_bus_cat2', overwrite=True, cnames=True)
    #tb
    grass.run_command('v.out.ogr',input='iso_streets_tb',output=inter_path, format='ESRI_Shapefile',overwrite=True)
    path_streets_tb_cat=os.path.join(inter_path, 'iso_streets_tb.shp')
    grass.run_command('v.in.ogr', input=path_streets_tb_cat, output='streets_tb_cat2', overwrite=True, cnames=True)
```

Figure 14: Converting net layers

For the time needed to get from any street to the closest stop, a new column is created in the attribute table of the feature which already contains the correct value (fig.15). This has to be done for buses, trams and a mixed route between both transportation methods.

```
#adds cost value from center to stop in new column

    #tram

    grass.run_command('v.db.addcolumn', map='streets_tram_cat2', columns='first_tram_distance2 int',overwrite=True)

    grass.run_command('v.distance',from_='streets_tram_cat2', to='tramstop', upload='to_attr', to_column='first_tram_distance',column='first_tram_distance2',
overwrite=True)

    #bus

    grass.run_command('v.db.addcolumn', map='streets_bus_cat2', columns='first_bus_distance2 int',overwrite=True)

    grass.run_command('v.distance',from_='streets_bus_cat2', to='busstop', upload='to_attr', to_column='first_bus_distance',column='first_bus_distance2',
overwrite=True)

    #tb

    grass.run_command('v.db.addcolumn', map='streets_tb_cat2', columns='first_tb_distance2 int',overwrite=True)

    grass.run_command('v.distance',from_='streets_tb_cat2', to='tbstop', upload='to_attr', to_column='first_tb_distance',column='first_tb_distance2', overwrite=True)
```

Figure 15: New column for time needed between streets and stops

A final column is added, filled with the time needed to reach the closest stop plus the necessary time to travel from that stop to the center (fig.16). For a mixed net containing trams and buses (tb), '+5' is added to simulate at least one switch in lines between bus and tram lines. Obviously the 5min change time added this way is a rough estimation since the average waiting time can vary for different cities, weekdays and times of day. For our study area Heidelberg and Karlsruhe most buses and trams arrive on a 10 minute rhythm between Monday and Friday, which would result in an average waiting time around 5 minutes.

```
#adds both values -> final time cost from street to center without waiting times
    #tram
    grass.run_command('v.db.addcolumn', map='streets_tram_cat2', columns='final_costs_tram int',overwrite=True)
    grass.run_command('v.db.update', map='streets_tram_cat2',layer=1, column='final_costs_tram', query_column="first_tram_distance2 + cat_",overwrite=True)
    #bus
    grass.run_command('v.db.addcolumn', map='streets_bus_cat2', columns='final_costs_bus int',overwrite=True)
    grass.run_command('v.db.update', map='streets_bus_cat2',layer=1, column='final_costs_bus', query_column="first_bus_distance2 + cat_",overwrite=True)
    #tb !!+5 because you need to wait between lines
    grass.run_command('v.db.addcolumn', map='streets_tb_cat2', columns='final_costs_tb int',overwrite=True)
    grass.run_command('v.db.update', map='streets_tb_cat2',layer=1, column='final_costs_tb', query_column="first_tb_distance2 + cat_ + 5",overwrite=True)
#deleting lines not connected to the center
    grass.run_command('v.db.update', map='streets_bus_cat2', column='final_costs_bus', value=99999, where="first_bus_distance2 = 0",overwrite=True)
```

Figure 16: Total travel period & change of lines

In preparation for the last step of our analysis we add the final cost values for the bus and tram nets into the mixed net 'streets_tb_cat2' for easier comparability (fig.17).

```
#adding all final colums into streets_tb_cat2 for easier comparability
    grass.run_command('v.db.addcolumn', map='streets_tb_cat2', columns='tram_cost int',overwrite=True)
    grass.run_command('v.distance',from_='streets_tb_cat2', to='streets_tram_cat2', upload='to_attr', to_column='final_costs_tram',column='tram_cost', overwrite=True)
    grass.run_command('v.db.addcolumn', map='streets_tb_cat2', columns='bus_cost int',overwrite=True)
    grass.run_command('v.distance',from_='streets_tb_cat2', to='streets_bus_cat2', upload='to_attr', to_column='final_costs_bus',column='bus_cost', overwrite=True)
```

Figure 17: Adding values into layer 'streets_tb_cat2'

The lowest value for each section on the streets is then calculated. The resulting values are a rough estimation of how much time it takes to reach the civic center from any given point in the city (fig.18)

```
#selecting the lowest possible value for each street
    grass.run_command('v.db.addcolumn', map='streets_tb_cat2', columns='lowest int',overwrite=True)
    grass.run_command('v.db.update', map='streets_tb_cat2',layer=1, column='lowest', qcolumn="tram_cost",overwrite=True, where="tram_cost <=
bus_cost<=final_costs_tb")
    grass.run_command('v.db.update', map='streets_tb_cat2',layer=1, column='lowest', qcolumn="bus_cost",overwrite=True, where="bus_cost <= tram_cost AND bus_cost <=
final_costs_tb")
    grass.run_command('v.db.update', map='streets_tb_cat2',layer=1, column='lowest', qcolumn="final_costs_tb", overwrite=True, where="final_costs_tb <= tram_cost AND
final_costs_tb <= bus_cost")
```

Figure 18: Calculating lowest value for each section

VI. Exporting final layers

By that the script is already finished with the analysis (fig.19). In order to be able to use the obtained data from the script outside of GRASS GIS, we export the vector map layer to end our script (fig.20). For example, the shapefiles could be used to visualize the results of the analysis in QGIS.

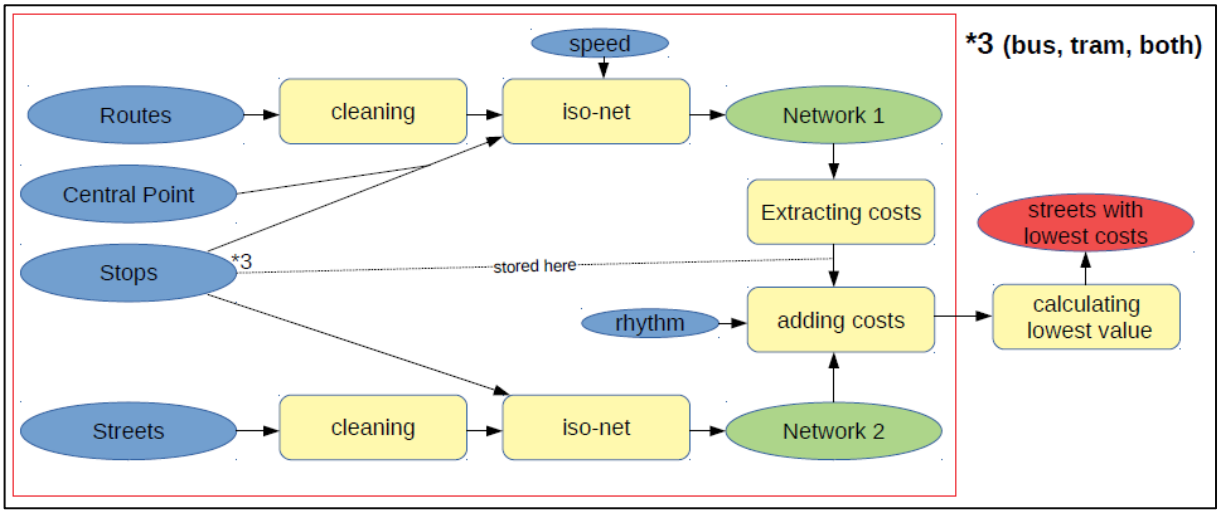


Figure 19: Complete workflow of the analysis

```
200  #exporting final layers^M
201      grass.run_command('v.out.ogr',input='streets_tb_cat2',output=out_path, format='ESRI_Shapefile',overwrite=True)^M
202      grass.run_command('v.out.ogr',input='streets_tram_cat2',output=out_path, format='ESRI_Shapefile',overwrite=True)^M
203      grass.run_command('v.out.ogr',input='streets_bus_cat2',output=out_path, format='ESRI_Shapefile',overwrite=True)^M
```

Figure 20: Export of obtained results

4. Visualized analysis results

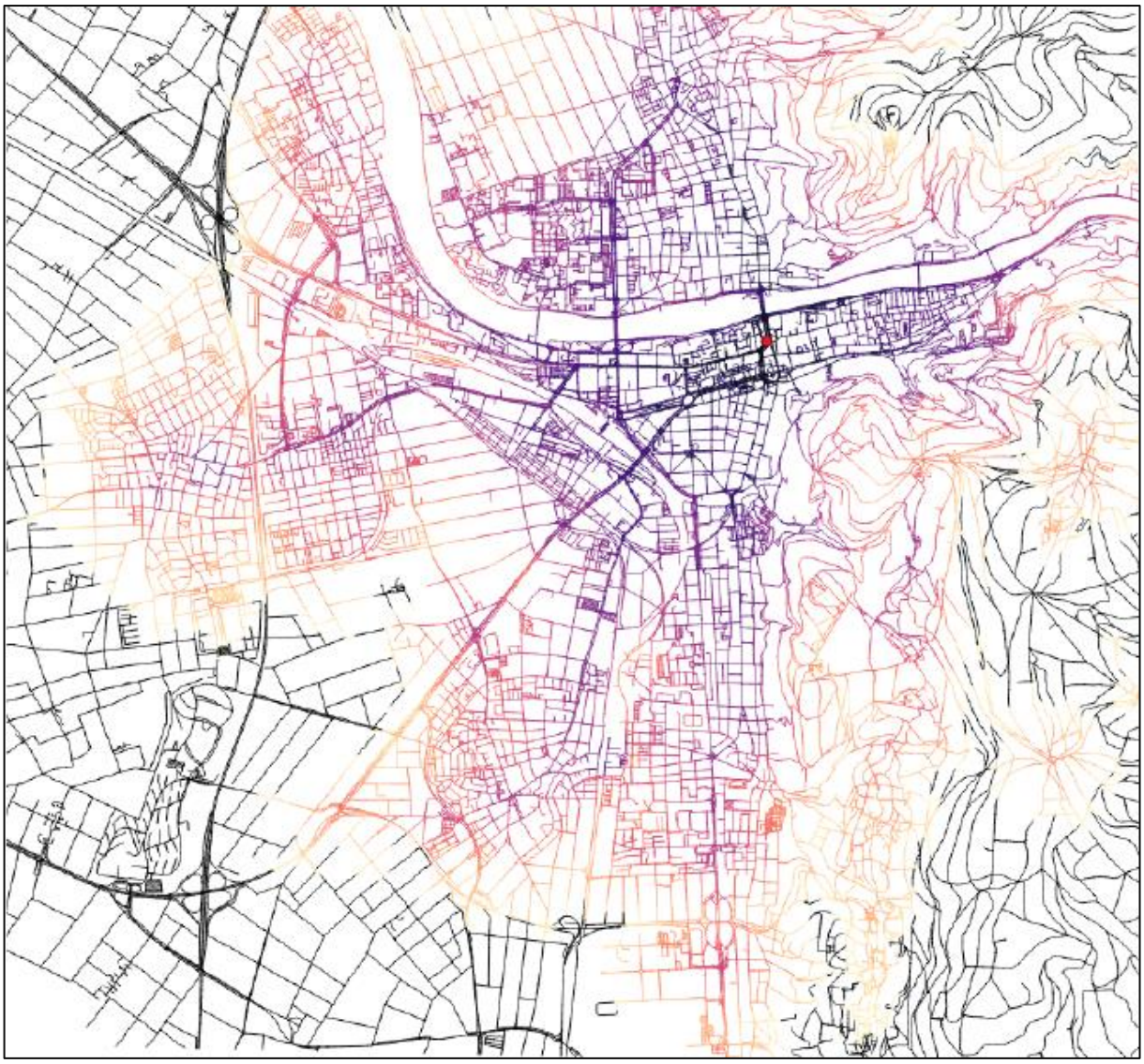


Figure 21: Visualized results of Heidelberg

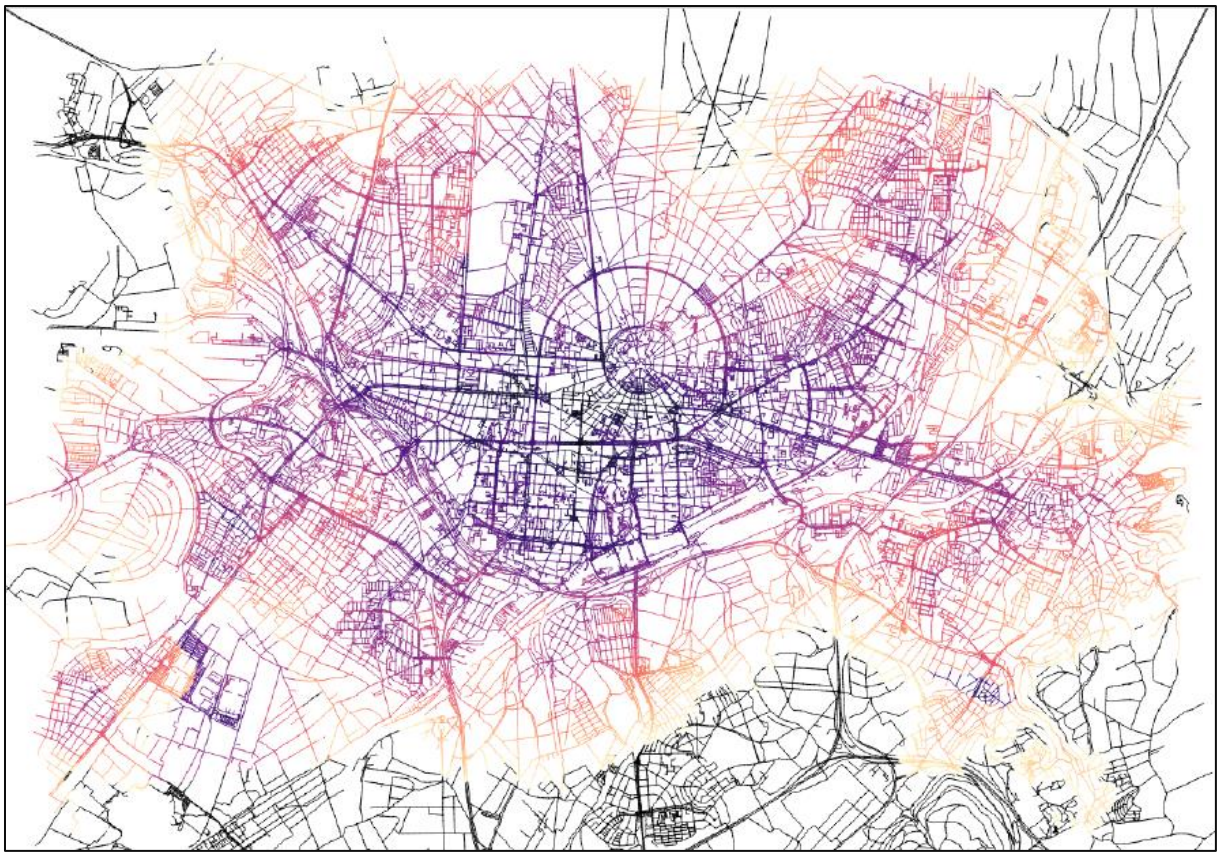


Figure 22: Visualized results of Karlsruhe

5. Solved issues

During the processing of our analysis we encountered various issues, which we already solved on our own:

Before we could import shapefiles into GRASS GIS it was necessary to change a few attribute names in tables, which are reserved key words in SQL – in this analysis ‘to’ and ‘from’.

The command ‘grass.run_command('v.distance',from_='tramstop', to='iso_tram', upload='cat', column='first_tram_distance', overwrite=True)’ caused an error since it uses a reserved keyword from SQL (‘from’). If you use an underscore after the keyword, the tool will work, e.g. ‘from_’.

In GRASS networks don’t work as individual features, for which reason the net had to be split. By exporting the net as shapefile and re-importing it again we fixed this problem. The same solution was used to split multiple features which shared the same ID.

6. Evaluation

As conclusion we will discuss the results of our analysis, reflecting on possible improvements and critique towards the script:

The script was tested on the study area of Heidelberg and Karlsruhe and worked accordingly, delivering pleasing results. For example the values given for Heidelberg in the areas of Eppelheim and Rohrbach were quite accurate compared to the actual travel times. However the script took between 4 to 15 minutes to process the original data. This might indicate that the algorithm used wasn't the most efficient.

The implementation for switching lanes isn't ideal. The static value added (+5 min) doesn't reflect the reality adequately, since public transportation schedules can vary heavily between different cities, week days or times of day. In rural areas an average of 5 minutes for a change of lines would be unrealistic. Additional the average speed of bus and trams may vary between different cities or countries too, which would require a manual change of the average speed times included in the analysis script. Instead of average speed times it would be possible to include actual time tables for the respective stops, which would lead to more accurate results especially in the area close to the civic center.

With a growing size of the analyzed cities and their public transportation network, the solution of the script gets increasingly worse. Since the algorithm always takes the shortest path towards the civic center, the chance of selecting a path which isn't actually used as public transportation line is increased. In very large cities the definition of one civic center might be difficult too, since many of those cities possess more than one major town square which would be suitable as civic center for the script. On top of that alternative transportation methods such as bike or train might have an increased relevance in bigger cities, which would promote an expansion of the script.