

Exercises: Power of Topology

Michael Wagner (mwagner@allspatial.info)

July 15, 2016

Contents

1	Introduction	4
2	Data preparation	4
3	Shortest path analysis with the QGIS Road Graph plugin	5
4	Shortest path analysis with pgRouting	9
5	Identifying service areas	12

List of Figures

1	Widget for shortest path analysis	6
2	Road Graph plugin configuration	7
3	Define start and destination for the shortest path analysis	7
4	Analysis result	8
5	Visualise road directions	8
6	Example result of shortest path analysis (bddijkstra)	11
7	Settings for <i>pgRouting Layer</i> plugin	12
8	Identifying nearest node to station	13
9	Settings to generate the cost surface	15
10	Settings to generate the isochrones	15
11	Clipping the cost surface	16
12	Clipping the isochrones	16
13	Styling the cost surface	17
14	Final map with isochrones, roads, fire stations and cost surface	18

List of Listings

1	Deleting columns	4
2	Renaming a column	4

3	Changing a column's data type	5
4	Deleting data from a table	5
5	Loading pgRouting extension	9
6	Setting columns to NULL	9
7	Building topology for the road data	9
8	Calculating cost values	10
9	Granting SELECT on node table	10
10	Shortest path analysis with bddijkstra	10
11	Create a spatial view to visualise the analysis results	11
12	Calculating travelling costs	13
13	Extracting minimum travelling costs	14

List of Abbreviations

SFRSA Seychelles Fire and Rescue Services Agency

1 Introduction

In this exercise you will do various analyses based on topology:

- Shortest path analysis with the QGIS Road Graph plugin
- Shortest path analysis using pgRouting, a routing extension for PostgreSQL/-PostGIS
- Generating a map of service areas that will show in what time the [Seychelles Fire and Rescue Services Agency \(SFRSA\)](#) can reach what areas

2 Data preparation

For the road network you will make use of the *road* table from your database *my_first_geodb* that you created yesterday. This table needs a little bit of fixing and cleaning before we will start with any analysis. In *pgAdmin* connect to database *my_first_geodb* and open a new SQL query dialog. We will first delete a number of columns that are not needed:

Listing 1: Deleting columns

```
1 ALTER TABLE vector_data.road DROP COLUMN numberofla;  
2 ALTER TABLE vector_data.road DROP COLUMN pavementy;  
3 ALTER TABLE vector_data.road DROP COLUMN island;  
4 ALTER TABLE vector_data.road DROP COLUMN cost_fp;  
5 ALTER TABLE vector_data.road DROP COLUMN reverse_01;
```

Then rename the column *reverse_co* to *reverse_cost*. Because of the Shapefile limitation of 10 characters for field names the original field name was cut off.

Listing 2: Renaming a column

```
1 ALTER TABLE vector_data.road RENAME COLUMN reverse_co TO  
reverse_cost;
```

During the Shapefile import all fields of data type integer were converted to float types. We will fix that and change their data type back to integer:

Listing 3: Changing a column's data type

```
1 ALTER TABLE vector_data.road ALTER direction TYPE int4;  
2 ALTER TABLE vector_data.road ALTER category TYPE int4;  
3 ALTER TABLE vector_data.road ALTER speedlimit TYPE int4;  
4 ALTER TABLE vector_data.road ALTER condition TYPE int4;  
5 ALTER TABLE vector_data.road ALTER source TYPE int4;  
6 ALTER TABLE vector_data.road ALTER target TYPE int4;
```

Finally we will empty the table to load a topological clean dataset afterwards:

Listing 4: Deleting data from a table

```
1 DELETE FROM vector_data.road;
```

From within the SQL query dialog load the *road_data_cleaned.sql* script (folder *Data/RoadData*) and run it. This will populate the currently empty *road* table with topologically clean road data for Mahe.

3 Shortest path analysis with the QGIS Road Graph plugin

Launch QGIS and load the *road* layer from the *my_first_geodb* database. Load the aerial photo of Mahe from the *Data/AerialPhoto* folder. Open the *Manage and Install Plugins* dialog from the *Plugins* menu. Search for *Road Graph* and activate the plugin. A new dock widget titled *Shortest path* should appear below the table of contents (Figure 1).

From menu *Vector:Road graph* open the *Settings* dialog and set the parameters as shown in Figure 2.

The topology tolerance is set to 0.1 (10cm). Any two line ends within 10cm distance are considered to be connected to the same node. For a road network five to ten centimetres should be a reasonable tolerance value. The *road* table

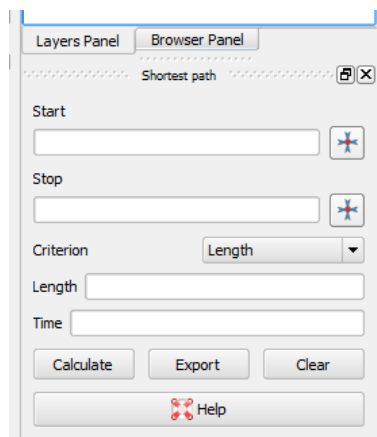


Figure 1: Widget for shortest path analysis

has a column with information about the direction (*direction*) and the speed limit (*speedlimit*). A value of 1 for direction means one-way road, a value of 2 stands for both directions. Zoom the map to somewhere near the cathedral in Victoria. In the *Shortest path* widget click the cross-hair button next to the *Start* field and click near the junction labelled with *F* in Figure 3. Click the cross-hair button next to the *Stop* field and click near the junction labelled with *T* in Figure 3. Finally click the *Calculate* button. The shortest path will be shown in the map and the time and distance calculated.

Now repeat the analysis but swap *Start* and *Stop* point before. This time you should get a different result. This is because some of the road segments are one-way only and thus, the shortest path is calculated accordingly (Figure 4).

In the *Shortest path* widget change the *Criterion* field from *Length* to *Time* and click the *Calculate* button again. The result will be the same as before since with the currently defined speed limits no other path would save more time. Now edit the *road* layer and reduce the speed limit on *Quincy Street* from 50 to 5 km/h. Repeat the analysis by clicking the *Calculate* button again. You should get a different result this time.

Sometimes it is useful to visualise the default direction of the road segments. You can do that by adding a *Marker Line* to your simple line currently used for the *road* layer style. The result might look like in Figure 5.

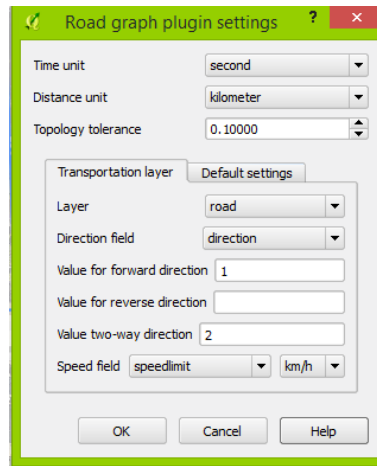


Figure 2: Road Graph plugin configuration



Figure 3: Define start and destination for the shortest path analysis

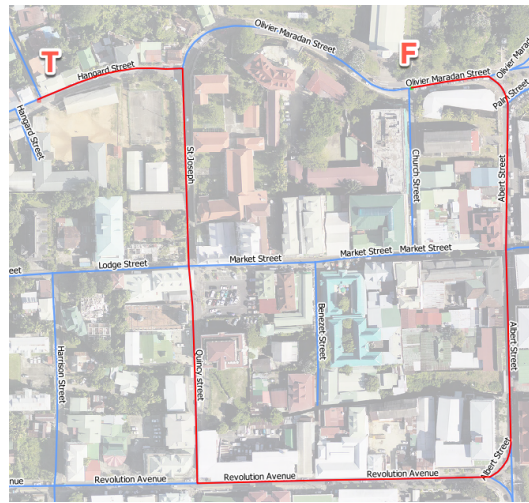


Figure 4: Analysis result

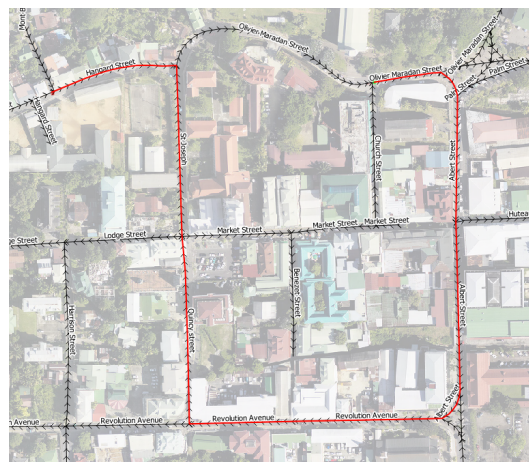


Figure 5: Visualise road directions

4 Shortest path analysis with pgRouting

pgRouting is an extension for PostgreSQL/PostGIS to add routing functionality on database level. Thus, the routing engine is basically the database. The advantage is that all calculation is done on the server (which would often have more computing power than a laptop or desktop). Different client applications (e.g. QGIS or a web or mobile application) then just need to visualise the results. *pgRouting* requires PostGIS to be installed and loaded in the database to use *pgRouting* with. Your database *my_first_geodb* has the PostGIS extension already loaded. What is left is to load the *pgRouting* extension itself. Loading an extension can only be done by a database superuser. Using *pgAdmin* connect to database *my_first_geodb* as user *postgres* and open an SQL query dialog. Run the following statement to load the *pgRouting* extension:

Listing 5: Loading *pgRouting* extension

```
1 CREATE EXTENSION pgrouting;
```

Open another SQL query dialog for *my_first_geodb*, this time as user *geodb_admin*. There is some more preparation we need to do before we can run a *pgRouting* analysis. We have to set the *source* and *target* columns to NULL because we want to build new topology in a moment.

Listing 6: Setting columns to NULL

```
1 UPDATE vector_data.road SET source = NULL, target = NULL;
```

Next, we will create proper topology for our road data. *pgRouting* provides a helper function to do that. This function will first create a new table named *road_vertices_pgr* and populate it with the nodes. *pgRouting* will then populate the *source* and *target* columns of our *road* table with the IDs of the relating nodes. The function is called like so:

Listing 7: Building topology for the road data

```
1 SELECT pgr_createTopology('vector_data.road', 0.1, 'geometry', 'gid  
    ');
```

The function expects the follows parameters:

- Schema and table name of the edge table
- Tolerance (0.1m)
- Name of the geometry column
- Name of the primary key column

Now we will populate the cost columns of the *road* table. The costs are crucial because based on the costs pgRouting will calculate the shortest path. We will use length and speed limit of the road segments to calculate the costs as time in minutes needed to travel a segment. We set the *reverse_cost* to same value as *cost* and set *reverse_cost* to -1 for all one-way roads.

Listing 8: Calculating cost values

```
1 UPDATE vector_data.road SET cost = ((ST_Length(geometry)/1000)/  
   speedlimit)*60;  
2 UPDATE vector_data.road SET reverse_cost = cost;  
3 UPDATE vector_data.road SET reverse_cost = -1 WHERE direction = 1;
```

Next, grant a SELECT privilege on the node table:

Listing 9: Granting SELECT on node table

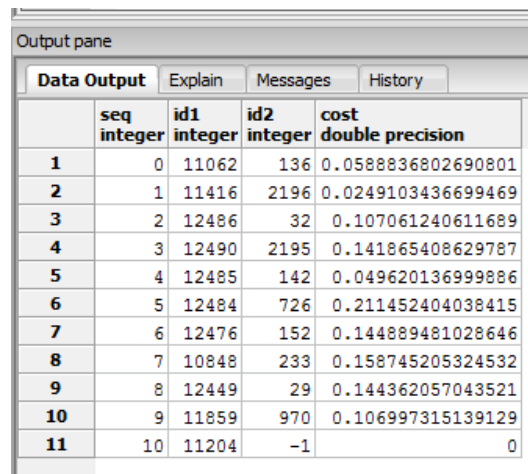
```
1 GRANT SELECT ON vector_data.road_vertices_pgr TO gis_update,  
   gis_view;
```

Load the node table (*road_vertices_pgr*) in QGIS and label it with the node IDs. Write down the node IDs of two nodes that you want to find the shortest path between. Run the following query to perform the shortest path analysis using the *Bidirectional Dijkstra-Algorithm* and replace 281 and 500 with the IDs of the nodes that you chose as start and end point of your analysis.

Listing 10: Shortest path analysis with bddijkstra

```
1 SELECT * FROM pgr_bddijkstra('SELECT gid::int4 AS id, source::int4,
    target::int4, cost::float8, reverse_cost::float8 FROM
    vector_data.road', 281, 500, true, true);
```

Figure 6 shows an example result of the shortest path analysis using the *Bidirectional Dijkstra Algorithm*. Column *id1* contains the IDs of the nodes while column *id2* contains the IDs of the edges. The *cost* column contains the cost (in minutes) to travel the particular edge.



	seq integer	id1 integer	id2 integer	cost double precision
1	0	11062	136	0.0588836802690801
2	1	11416	2196	0.0249103436699469
3	2	12486	32	0.107061240611689
4	3	12490	2195	0.141865408629787
5	4	12485	142	0.049620136999886
6	5	12484	726	0.211452404038415
7	6	12476	152	0.144889481028646
8	7	10848	233	0.158745205324532
9	8	12449	29	0.144362057043521
10	9	11859	970	0.106997315139129
11	10	11204	-1	0

Figure 6: Example result of shortest path analysis (bddijkstra)

To visualise the shortest path in QGIS we can create a spatial view. As user *user_name* open an SQL query dialog for database *my_first_geodb* in pgAdmin. Run the query as follows to create the spatial view:

Listing 11: Create a spatial view to visualise the analysis results

```
1 CREATE OR REPLACE VIEW shortest_path AS
2 SELECT * FROM vector_data.road WHERE gid IN
3 (SELECT id2 FROM pgr_bddijkstra('SELECT gid::int4 AS id, source::
    int4, target::int4, cost::float8, reverse_cost::float8 FROM
    vector_data.road', 281, 500, true, true));
```

Load this view into QGIS and select the *gid* as the Feature ID column. Creating a spatial view for each query result is not too convenient. It would be much better

to have a tool available similar to the Road Graph plugin. Fortunately there is such tool. From the *Manage and Install Plugins* dialog install the *pgRoutingLayer* plugin. This plugin supports visualising the results of most of the algorithms pgRouting provides. In the *pgRouting Layer* widget do the settings as shown in Figure 7.

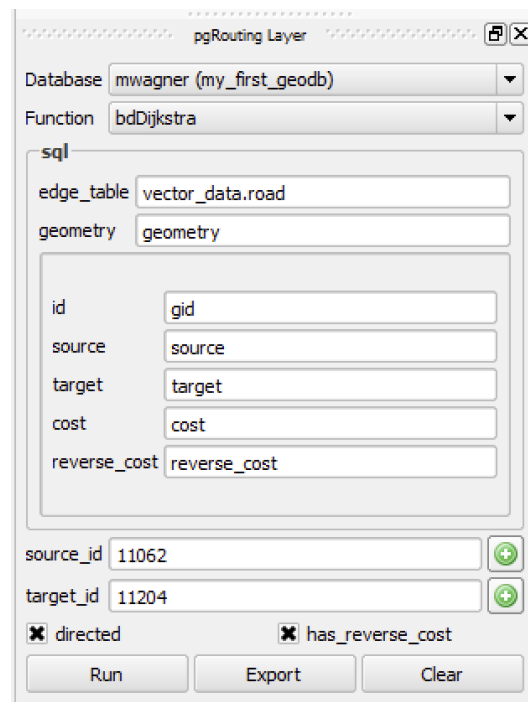


Figure 7: Settings for *pgRouting Layer* plugin

Set the source and target point in the map using the button next to the *source_id* and *target_id* field and click *Run*. pgRouting will calculate the shortest path and the plugin will visualise it in the map.

5 Identifying service areas

We want to find out in what time the **SFRSA** teams at the two fire stations on Mahe can reach what area. This type of analysis can help them improving their response times for example. Load the *fire_station* Shapefile from the *Data/Shapefiles* folder.

Visually identify the node that is nearest to each station and write down the ID of these nodes, e.g. the nearest node in Figure 8 would be 12276 (node in grey, station in green).



Figure 8: Identifying nearest node to station

Run the following SQL statement to calculate the costs (i.e. the time) for travelling from these two nodes to every other node in the road network and create a table with the results. Replace the IDs 11127 and 11292 with the IDs of the nodes that you identified as closest to the fire stations.

Listing 12: Calculating travelling costs

```

1 CREATE TABLE combined_driving_times AS
2 SELECT
3     id,
4     the_geom,
5     (select sum(cost) FROM
6     (
7     SELECT * FROM pgr_bddijkstra('SELECT gid::int4 as id, source::
        int4, target::int4, cost::float8, reverse_cost::float8 FROM
        vector_data.road', 11127, id::int4, TRUE, TRUE)) as foo) AS
        cost
8 FROM vector_data.road_vertices_pgr

```

```

9 UNION
10 SELECT
11     id,
12     the_geom,
13     (select sum(cost) FROM
14     (
15     SELECT * FROM pgr_bddijkstra('SELECT gid::int4 as id, source::
        int4, target::int4, cost::float8, reverse_cost::float8 FROM
        vector_data.road', 11292, id::int4, TRUE, TRUE)) as foo) AS
        cost
16 FROM vector_data.road_vertices_pgr;

```

The new table will have two travelling cost entries for each node but we are only interested in the one that is lowest. Run the statement as follows to create a new table with only the minimum costs.

Listing 13: Extracting minimum travelling costs

```

1 CREATE table min_driving_times AS
2 SELECT id, the_geom, min(cost) AS cost
3 FROM combined_driving_times
4 GROUP BY id, the_geom;

```

Load the *min_driving_times* table into QGIS. We will now create a cost surface by interpolating the costs from the *min_driving_times* table. In this cost surface each cell represents the travelling time to reach that cell. Select the *Interpolation* tool from the *Raster* menu. Apply the settings shown in Figure 9 and click *Ok*. Generating the cost surface might take a few seconds.

We can now generate isochrones (lines connecting points of the same time interval) from the cost surface. Select the *Contour* tool from the *Raster:Extraction* menu and apply the setting shown in Figure 10.

Finally we will clip the *cost_surface* and *isochrones* layers to the outline of Mahe. Load the *mahe_island* Shapefile into QGIS. From the *Raster:Extraction* menu select the *Clipper* tool and apply the setting as in Figure 11.

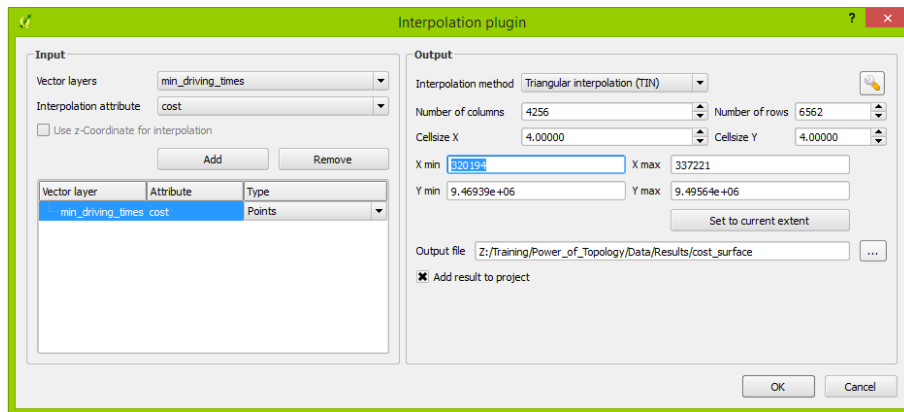


Figure 9: Settings to generate the cost surface

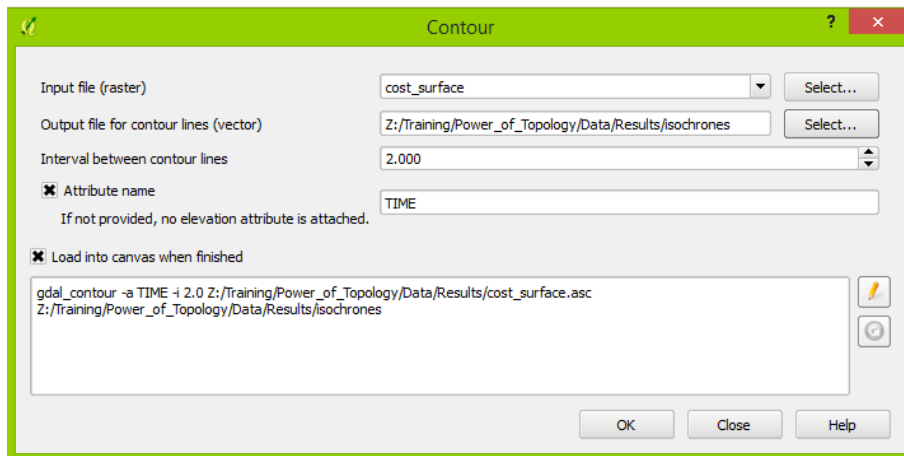


Figure 10: Settings to generate the isochrones

After the clipping remove the original *cost_surface* layer from the table of contents. To clip the *isochrones* use the *Clip* tool from the *Vector:Geoprocessing* menu (Figure 12).

Go to the style settings of the *cost_surface-clipped* layer and apply the settings shown in Figure 13. After clipping remove the original *isochrones* layer from the table of contents.

Finally label the *isochrones-clipped* layer with the values from the *TIME* field-/column. The final result map might look like Figure 14.

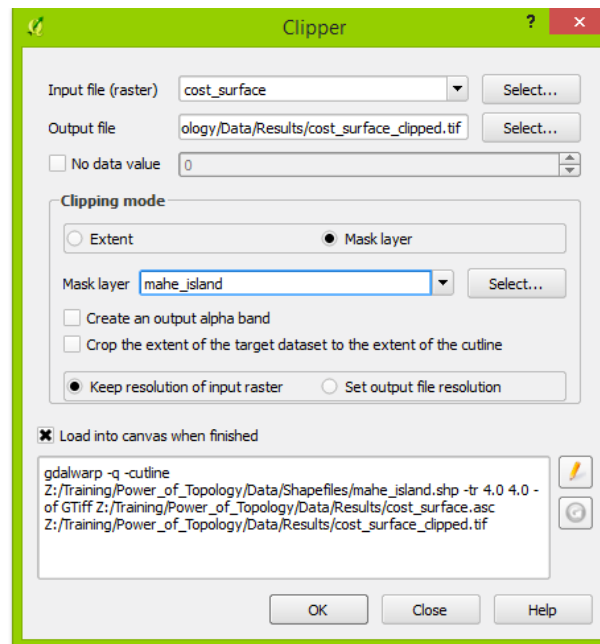


Figure 11: Clipping the cost surface

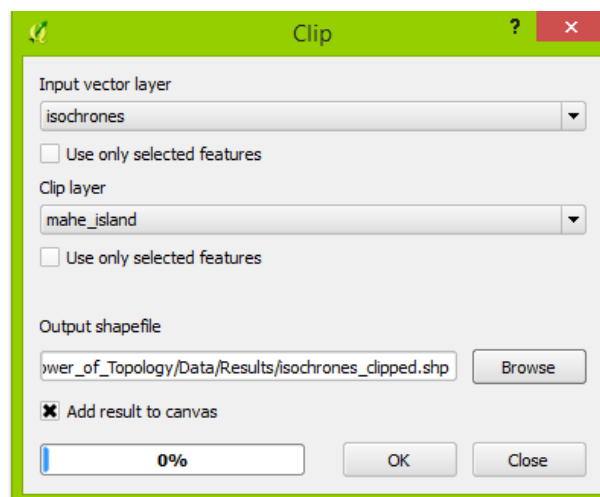


Figure 12: Clipping the isochrones

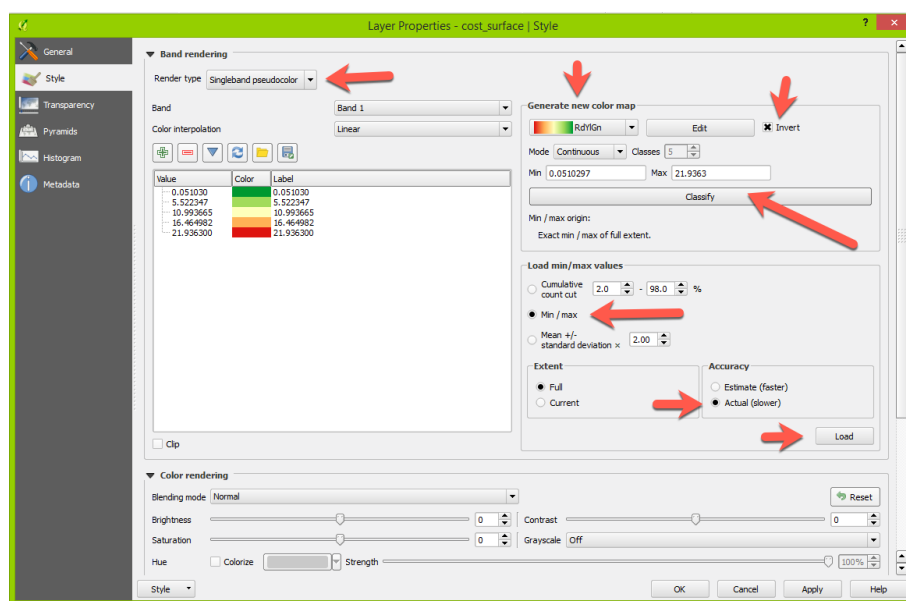


Figure 13: Styling the cost surface

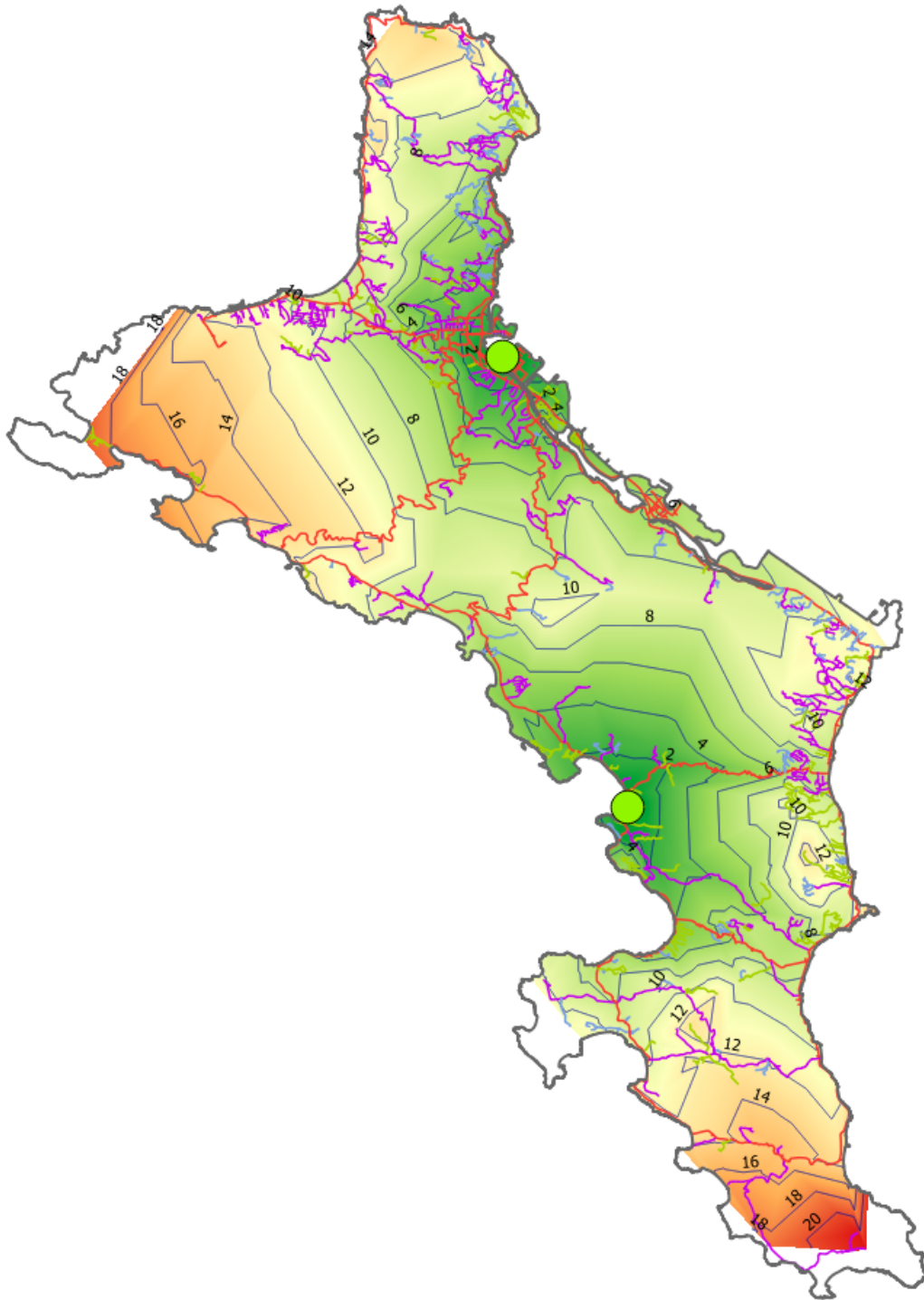


Figure 14: Final map with isochrones, roads, fire stations and cost surface