NTNU – Trondheim
Norwegian University of
Science and Technology

IT3105 - AI Programming

Project 3 Report

Self-Organizing Maps

Annie Aasen
Mikael Bjerga

November 6, 2016

1 Description of System

We first let our system define some global parameters used in the algorithm. These are which data set the algorithm should be performed on, the initial learning rate, the initial neighborhood radius, the number of neurons, the iteration limit, a constant used in the decay, the decay interval, the plot interval, and finally which type of decay should be used for the learning rate and radius. Some of these are initially set to none because they depend on the loaded data set. They will be set after actually loading the data set.

Our system then reads in the data from the specified travelling salesman problem (TSP) data set. This is done in a helper function and a text file is read into a two-dimensional numpy array where each row represent a city with the first column being the x-coordinate and the second column being the y-coordinate. The function returns one scaled version of the data as well as one unscaled version. The scaled version is scaled so that all coordinates are between the value 0 and 1, and this is done by dividing all coordinates by the corresponding maximum x- or y-coordinate.

Now that we have loaded the data set, we can calculate the number of neurons based on the number of cities in the TSP data set, and the initial neighborhood radius based on the number of neurons. We initialize a self-organizing-map (SOM) with random values between 0 and 1. The datastructure for the SOM is a numpy array with the same structure as the numpy array containing the coordinates of the cities. Each row here corresponds to a neuron, and we use the index of a row to refer to a specific neuron.

We continue on by using the SOM to create a scaffold that can be used to find a solution for the travelling salesman problem. This is where the learning algorithm is implemented. The algorithm performs a number of iterations based on the set iteration limit. For each iteration we pick a random city from the scaled city array. We find the best matching unit (BMU) of this city by comparing the distance from the chosen city to each neuron and selecting the closest one. The distance is computed using the formula for the Euclidean distance between two points in a two-dimensional space. Then we update the position of this neuron according to an update function, i.e. we move the neuron closer to the city. We also move the neighboring neurons according to the neighborhood radius a fraction closer depending on how far away they are from the BMU in the ring. Here it is important to note that the connections between the neurons form a cycle, so we need to use modulo operations to make sure the correct neighbors are also updated.

For a certain iteration interval we plot the scaffold as well as the intermediate solution and the total distance of the solution. For a certain decay interval we apply the chosen

type of decay to the learning rate and neighborhood radius. When the iteration limit is reached, the scaffold is returned and we can now use it to find a solution for the TSP. This is done by going through the scaled list of cities and matching each city with the nearest neuron of the scaffold, i.e. finding the BMU for the city. The matches are stored in a two-dimensional list where each inner list corresponds to a neuron and the entries of that list are all cities attached to that neuron. If there are no cities attached to a neuron, the list will be empty. In our implementation, we ignore the ordering between cities attached to the same neuron, following the advice from the compendium.

After traversing through all cities, we create the solution by adding the cities to a solution list in consecutive order of appearance in the previously list of matches. The list will be the final order of visiting the cities and is the solution of the TSP. We calculate the total length that this solution gives us and plot this together with the solution.

2 Experiences Tuning the System

The most critical parameter values we needed to tune to get the system to achieve an acceptable performance level was the number of neurons in the scaffold, the amount of iterations in the training phase, and the length of the interval between each application of learning rate and neighborhood radius decay, in addition to the parameters used in each decay function. Other parameters that were quite straightforward to select were the initial learning rate and the initial radius, which was set to 1 and one tenth of the number of neurons (as specified in the compendium), respectively. For the static decay function, however, these two parameters were chosen after experimentation.

The number of neurons needed to be tuned in order to achieve a scaffold that was able to capture the finer details of the distributions of cities, i.e. how to traverse clusters of closely positioned cities. For a low number of neurons, e.g. as many neurons as cities, we were not able to capture these details, and therefore produced a solution in which the clusters were not traversed efficiently. A higher number proved more successful in regard to the clusters without diminishing performance elsewhere. We settled on twice the number of cities as a good number of neurons, as it was able to traverse the clusters sufficiently well, and any higher number did not yield improvements.

The amount of iterations in the training phase and the length of the interval between applying decay was tuned together with the decay functions. Starting at 1000 iterations and a decay at every 100 iterations, we found that our scaffold moved around a lot and it seemed that each update was somewhat negating the effects of the previous updates. By increasing the amount of iterations and decreasing the decay interval, whilst also altering the decay functions to take the change into account, we experienced that the scaffold lost this erratic behaviour and was able to slowly capture more and more details in the later stages. The early movement was beneficial to move as many of the randomly positioned neurons as possible to the area where the cities were positioned. We settled on 2500 iterations and a decay at every 10 iterations. The combination of many iterations and frequent decay allowed a rapid decrease in scaffold movement with a long period of fine tuning in the latter stage for the linear and exponential decay functions.

As mentioned above, our decay functions were tuned together with the amount of iterations and decay interval. This was done to ensure that our initial values for learning rate and neighborhood radius were slowly decaying towards a favorable end point. This end point was discovered through experimentation to be in between 0.05 and 0.1 for the learning rate. Such a point was low enough to allow proper fine tuning of the scaffold. A lower point proved to be ineffective, as the scaffold would hardly learn. Applying the

same reduction to the neighborhood radius, reducing it to be in between 5% and 10% of the original radius, also proved effective. With this in mind, we designed the decay functions to decay towards these end points based on how many times the decay function would be called.

For the static decay functions, we did not need to design any explicit decay, but had to find good starting values for the learning rate and radius. We used experimentation to discover values for these two parameters, expecting that lower values be more efficient, as they are better for fine tuning of the scaffold. After discovering a decent combination of values for these parameters, we settled on a learning rate of 0.25 and neighborhood radius of 1/20 of the number of neurons. These values gave us the best results for the static decay function, and provided a compromise between the coarse tuning in the beginning and the fine tuning in the end.

The linear functions were harder to pinpoint and needed more experimentation to acquire a good constant. Because we wanted the learning rate and radius to decrease more rapidly in the beginning and less towards the end, we subtracted the product of remaining number of iterations and a constant. For the radius, we also multiplied this product with the initial radius, to scale the subtractions properly (this wasn't needed for the learning rate, as the initial value is 1). Through our experiments, we found a constant that would help decrease the learning rate and radius to an acceptable end point. To make our code as general as possible, this constant was made dependent on our iteration amount and an iteration multiple. This constant proved sufficient to our decay function.

Our exponential decay functions went through a similar process as the linear ones, where a constant was needed to reach an acceptable end point. We used experimentation, and found that a constant of 0.001 worked well. A higher number would lead to a more aggressive decline in both learning rate and radius at the early iterations. The scaffold would then lack coarse tuning and the finer tuning would not improve it much. A lower number would lead to a weaker decline, which prevents the scaffold from reaching the desired level of fine tuning.

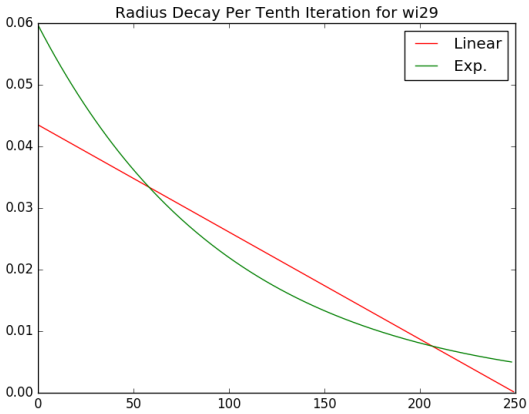
From our experience with these decay functions, the linear and exponential ones don't differ much in terms of results. Static decay performs the worst, which becomes more apparent in the later TSPs. Linear and exponential decay often give very similar results, although exponential decay seems to be more stable than the linear decay, which sometimes produce off results. Of the three, the static decay is the easiest to tune, but also the least adaptable, as we are not able to weight coarse tuning over fine tuning, or vice versa; the parameter values cannot change as time progresses. The other two are more adaptable in this sense, and the exponential function especially so. Of these two, we found the exponential decay function easier to tune, as any changes to the constant had less effect on the performance and therefore did not need the level of fine tuning the constant in the linear decay function needed.

3 Diagrams

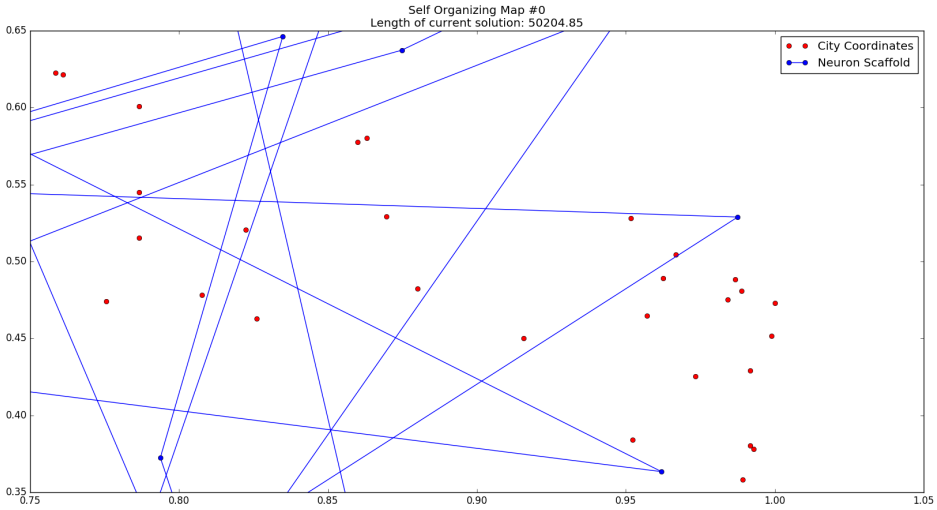
Western-Sahara

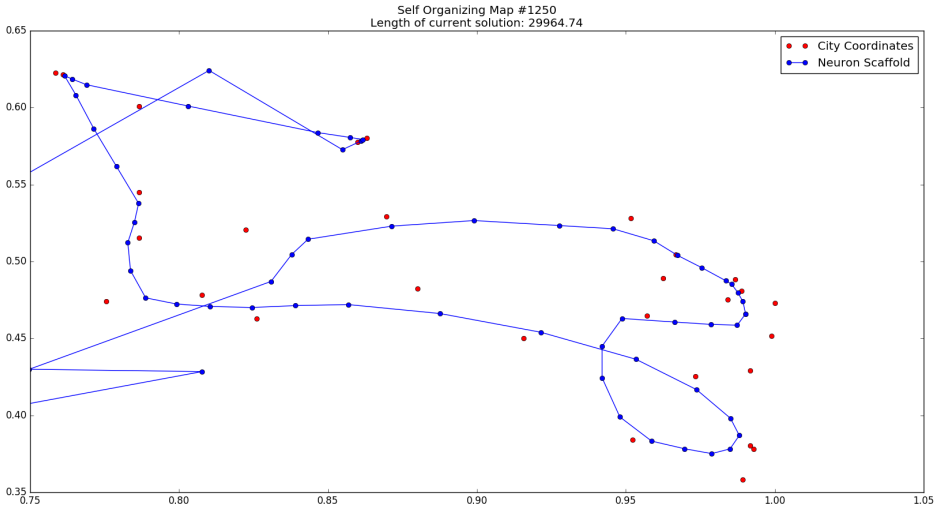
Diagrams for data set wi29 all use the same number of neurons, which is 58. The linear and exponential decay functions have an initial learning rate and initial neighborhood radius of 1 and 6, respectively. The static decay function has learning rate 0.25 and neighborhood radius 3.

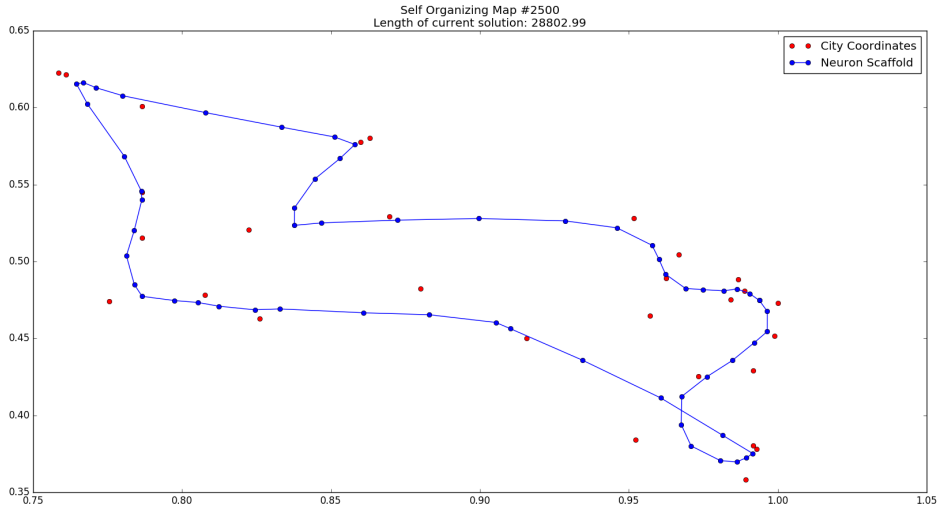
For the static decay function, there will be no change in learning rate or neighborhood size. The learning rate and neighborhood size change for the linear and exponential decay functions are shown in two separate plots:

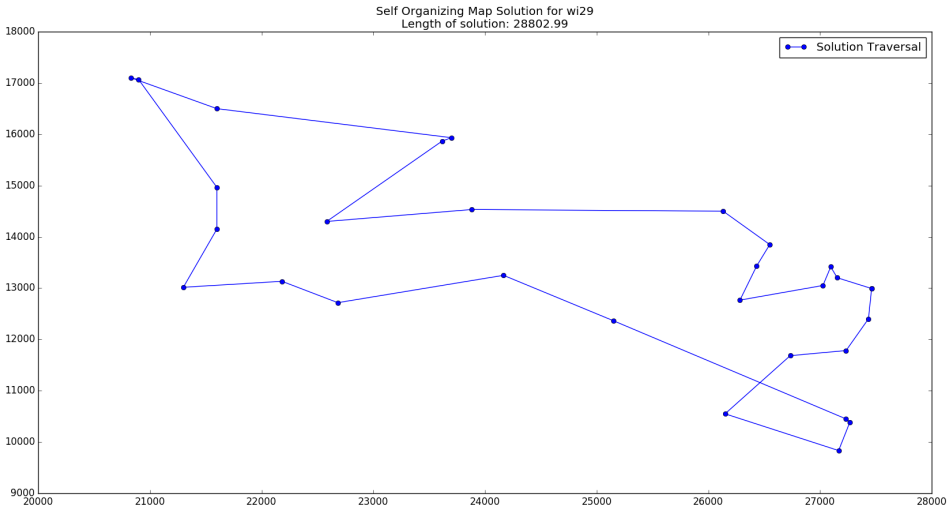


Best result using static decay function:

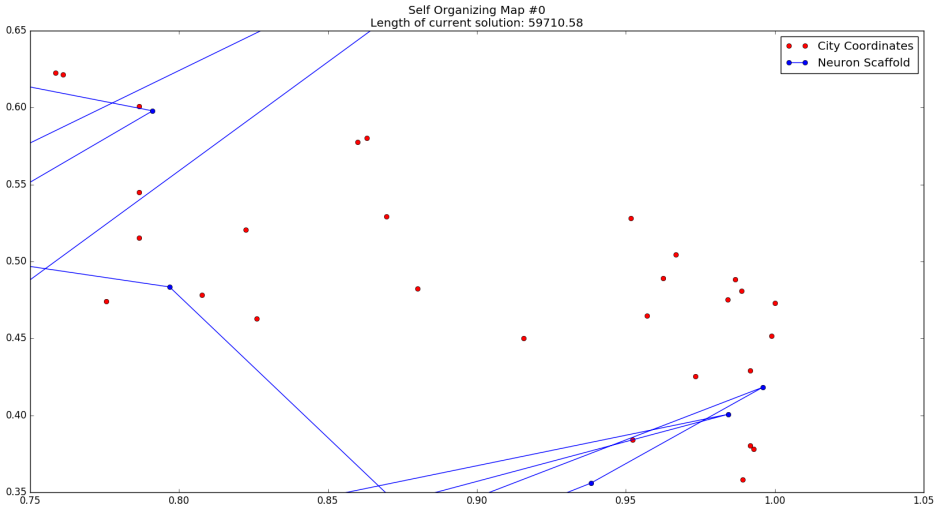


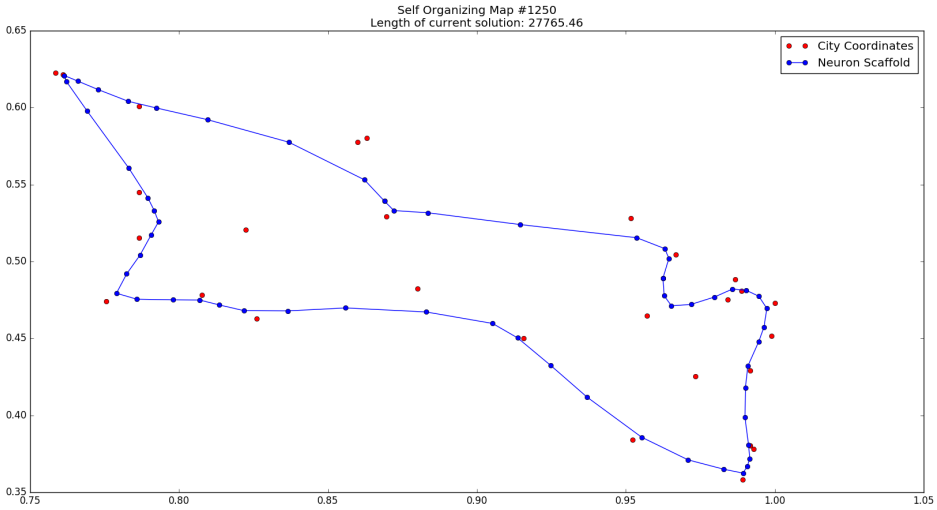


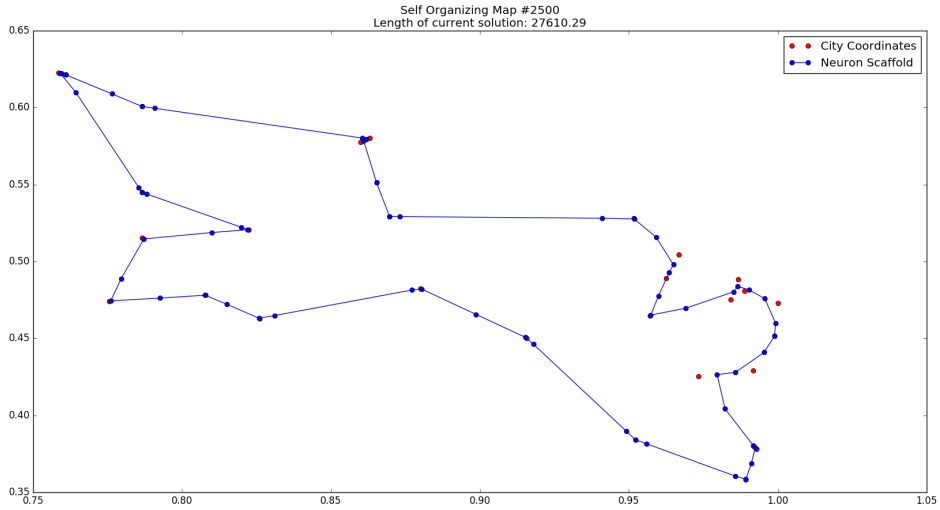

Self Organizing Map #2500
Length of current solution: 28802.99
City Coordinates
Neuron Scaffold
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.75
0.80
0.85
0.90
0.95
1.00
1.05

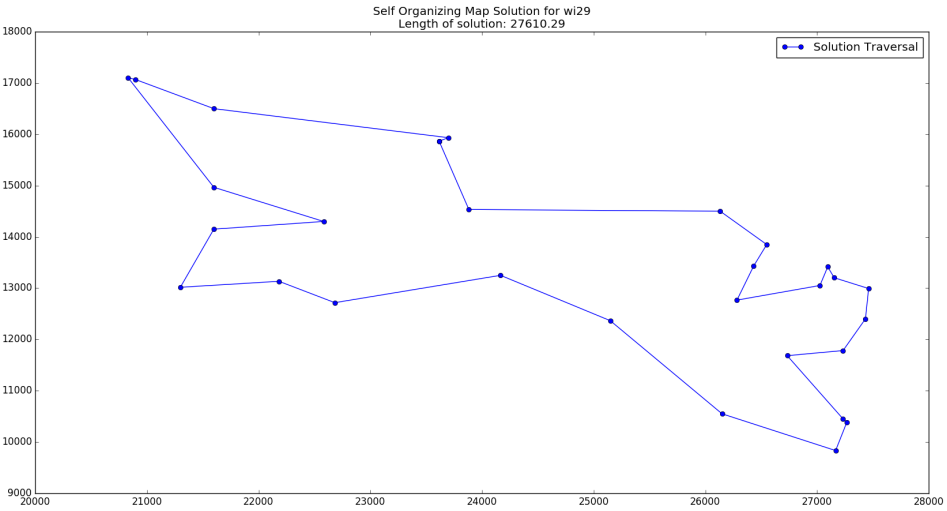

Self Organizing Map Solution for wi29
Length of solution: 28802.99
Solution Traversal
18000
17000
16000
15000
14000
13000
12000
11000
10000
9000
20000
21000
22000
23000
24000
25000
26000
27000
28000

Best result using linear decay function:

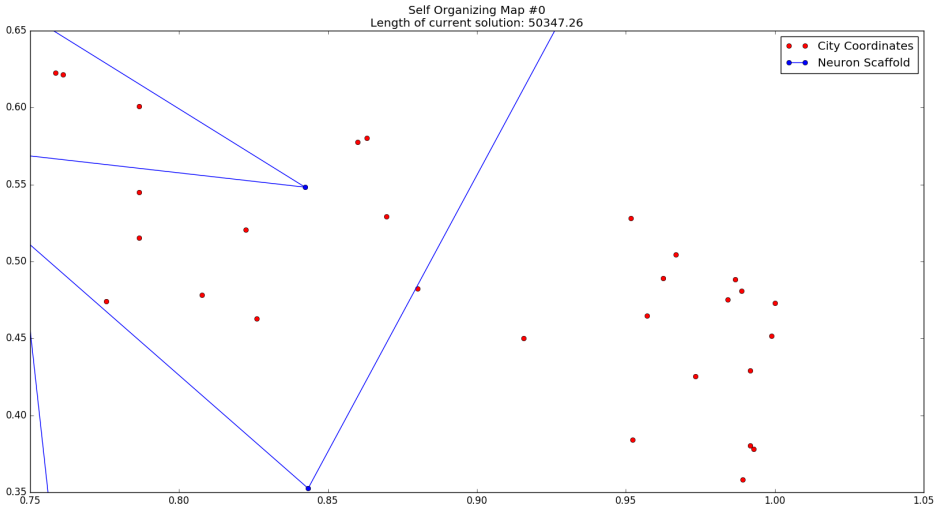


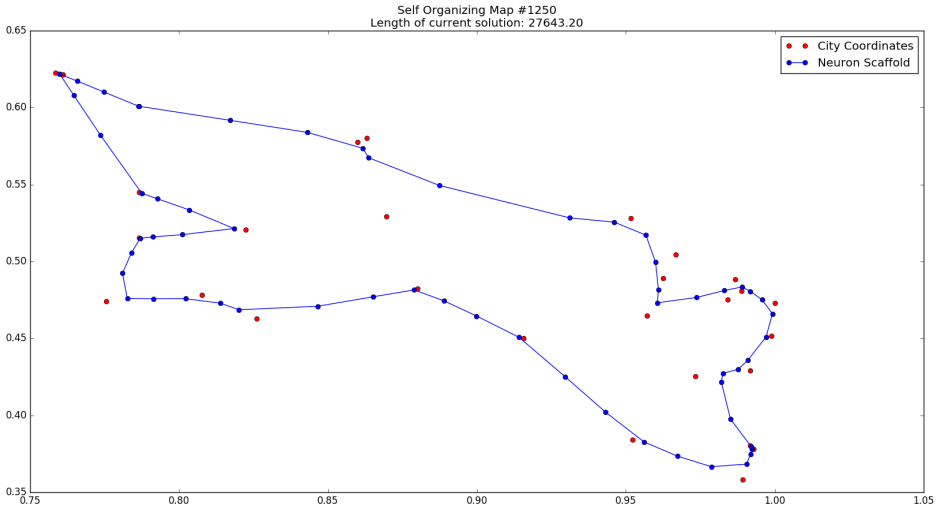


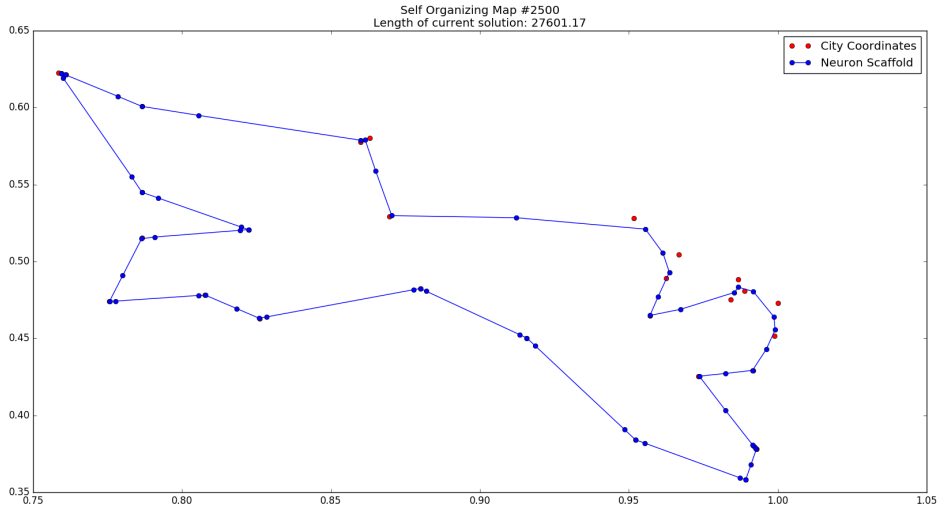

Self Organizing Map #2500
Length of current solution: 27610.29
City Coordinates
Neuron Scaffold
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.75
0.80
0.85
0.90
0.95
1.00
1.05

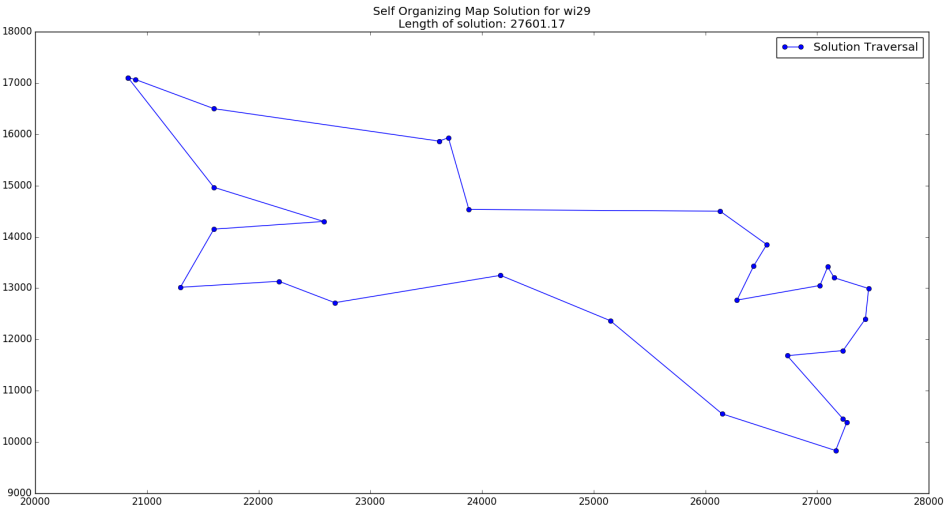

Self Organizing Map Solution for wi29
Length of solution: 27610.29
Solution Traversal
18000
17000
16000
15000
14000
13000
12000
11000
10000
9000
20000
21000
22000
23000
24000
25000
26000
27000
28000

Best result using exponential decay function:





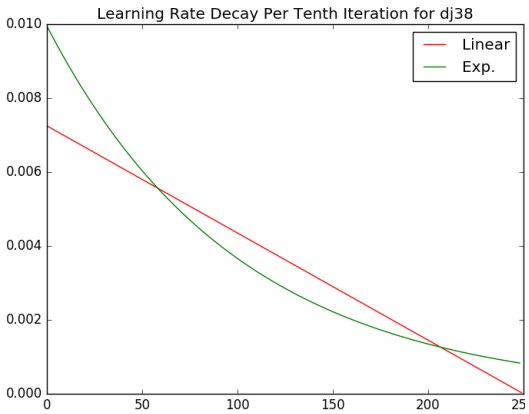

Self Organizing Map #2500
Length of current solution: 27601.17
City Coordinates
Neuron Scaffold
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.75
0.80
0.85
0.90
0.95
1.00
1.05

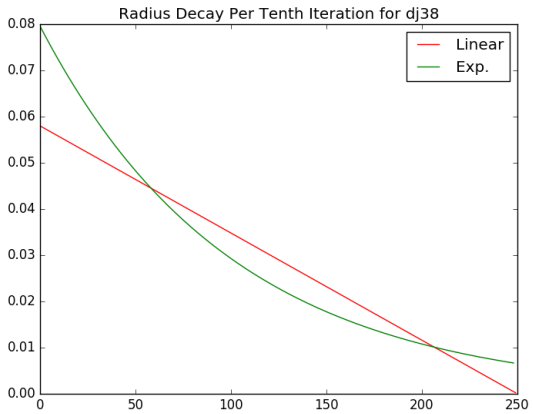

Self Organizing Map Solution for wi29
Length of solution: 27601.17
Solution Traversal
18000
17000
16000
15000
14000
13000
12000
11000
10000
9000
20000
21000
22000
23000
24000
25000
26000
27000
28000

Djibouti

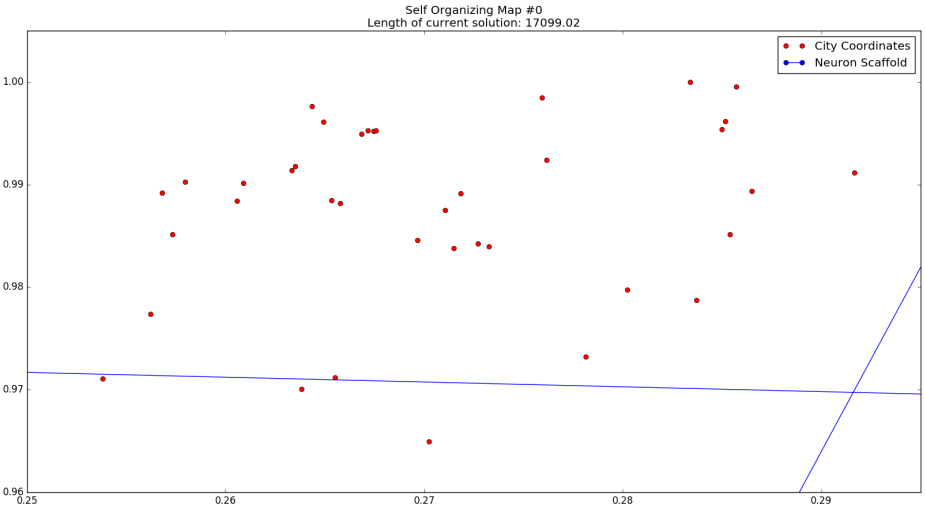
Diagrams for data set dj38 all use the same number of neurons, which is 76. The linear and exponential decay functions have an initial learning rate and initial neighborhood radius of 1 and 8, respectively. The static decay function has learning rate 0.25 and neighborhood radius 4.

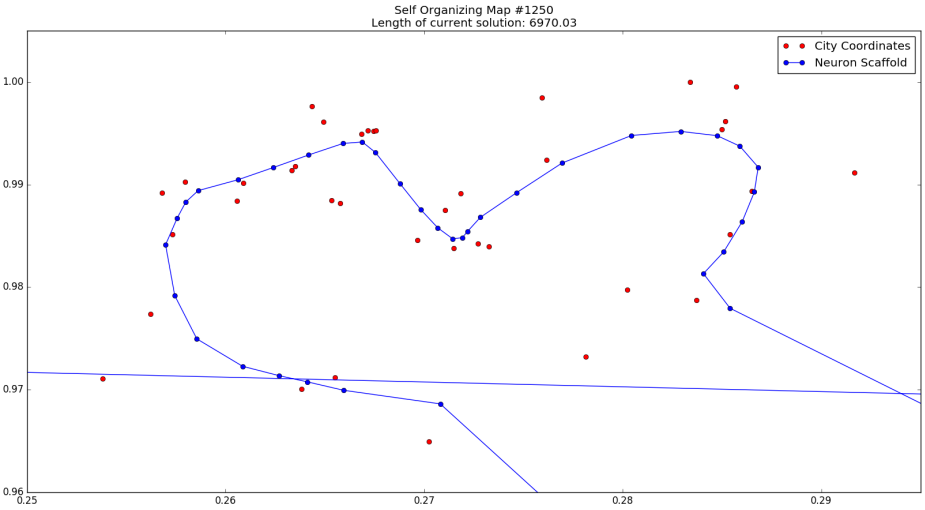
For the static decay function, there will be no change in learning rate or neighborhood size. The learning rate and neighborhood size change for the linear and exponential decay functions are shown in two separate plots:

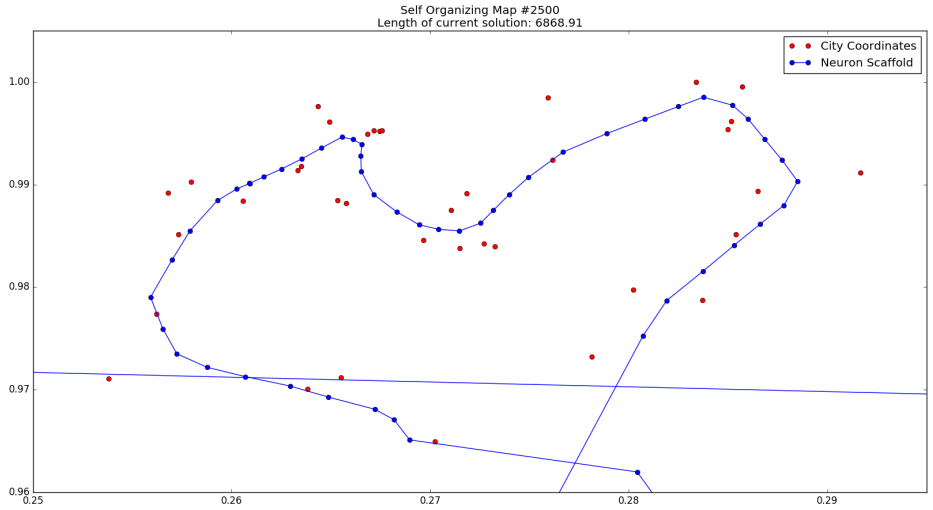


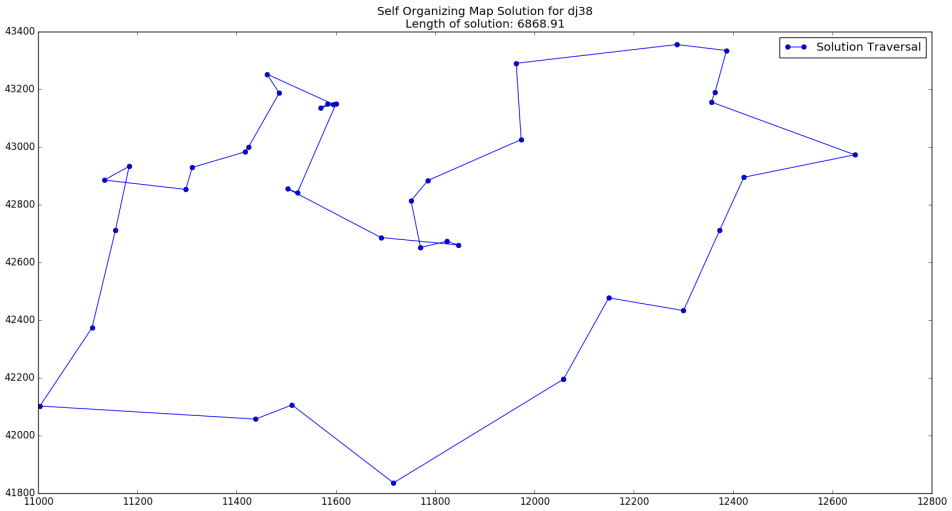


Best result using static decay function:

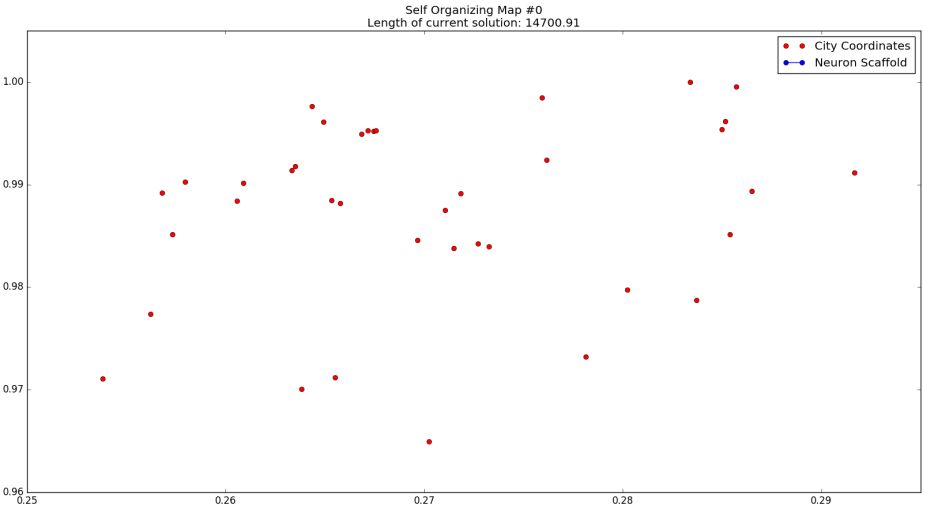


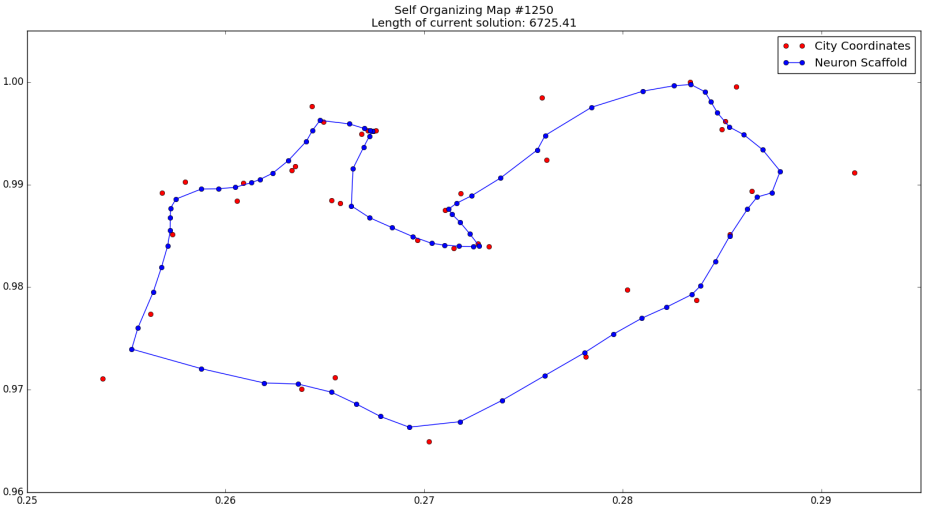


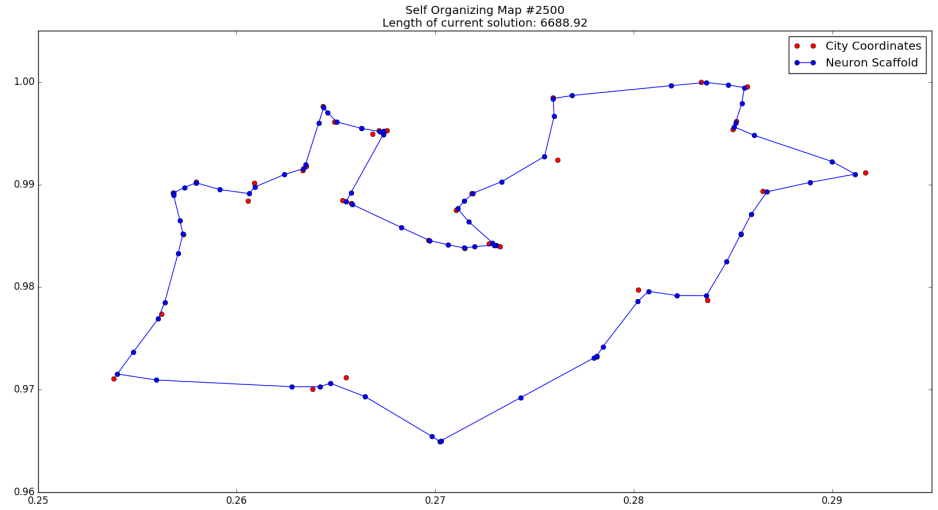

Self Organizing Map #2500
Length of current solution: 6868.91
City Coordinates
Neuron Scaffold
1.00
0.99
0.98
0.97
0.96
0.25
0.26
0.27
0.28
0.29

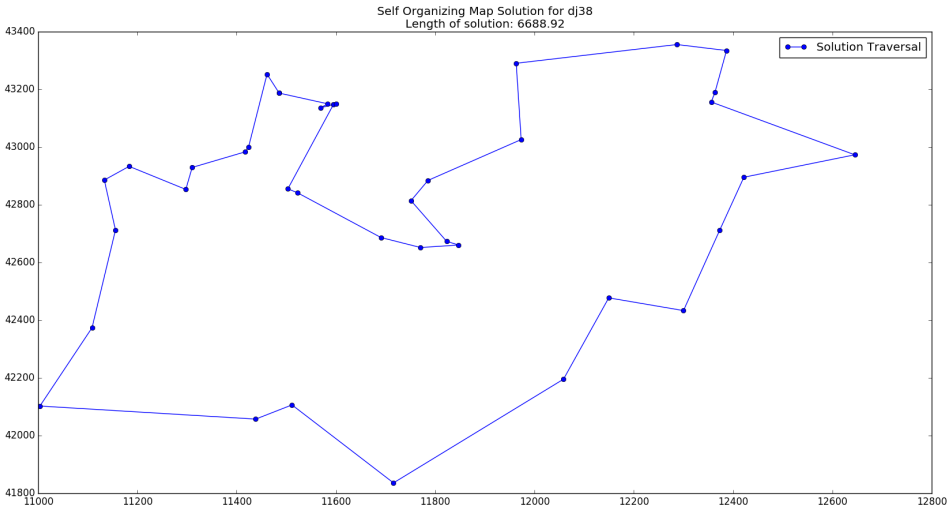

Self Organizing Map Solution for dj38
Length of solution: 6868.91
Solution Traversal
43400
43200
43000
42800
42600
42400
42200
42000
41800
11000
11200
11400
11600
11800
12000
12200
12400
12600
12800

Best result using linear decay function:

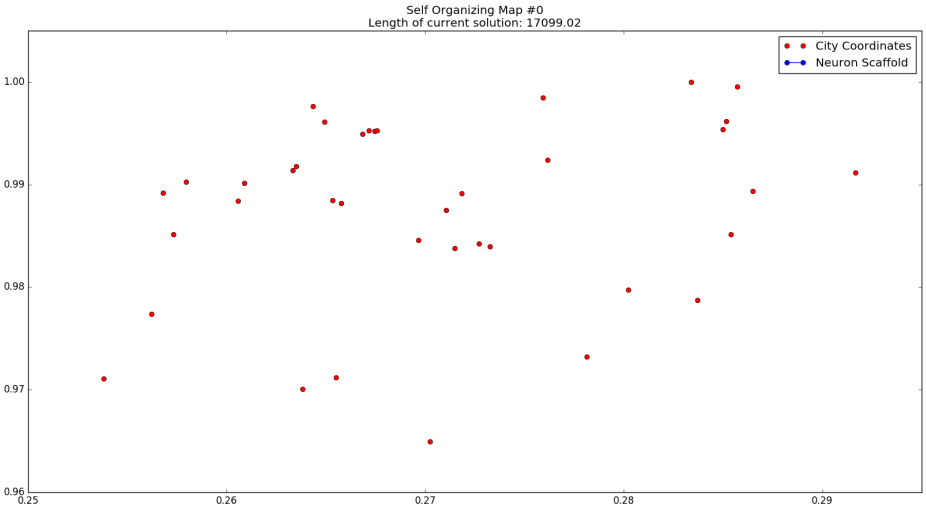


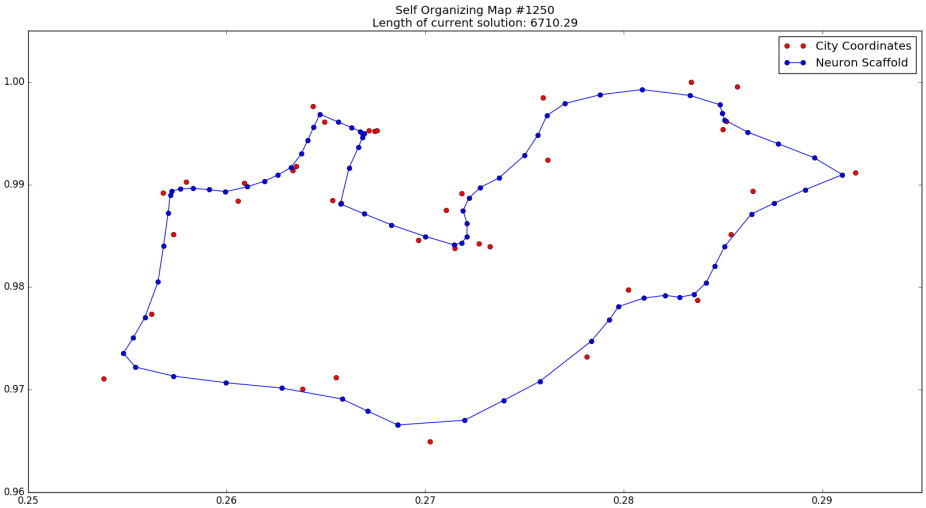


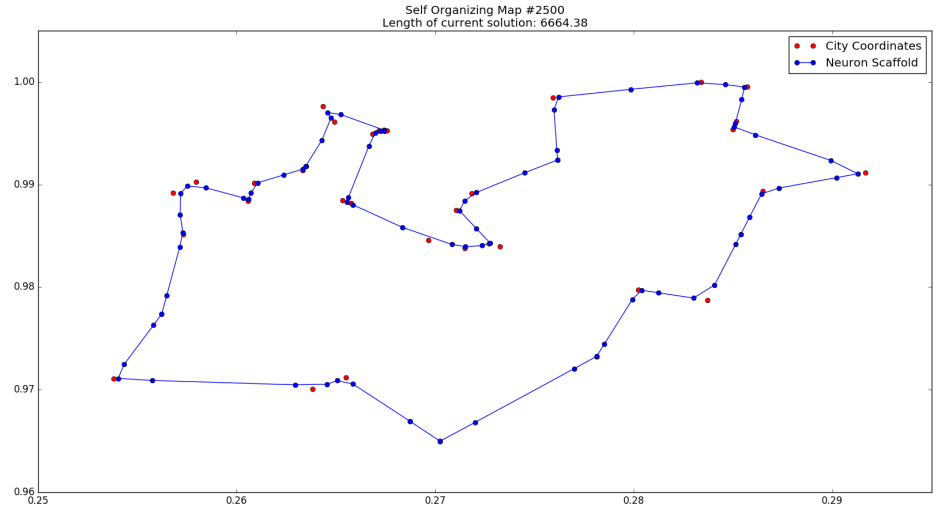

Self Organizing Map #2500
Length of current solution: 6688.92
City Coordinates
Neuron Scaffold
1.00
0.99
0.98
0.97
0.96
0.25
0.26
0.27
0.28
0.29

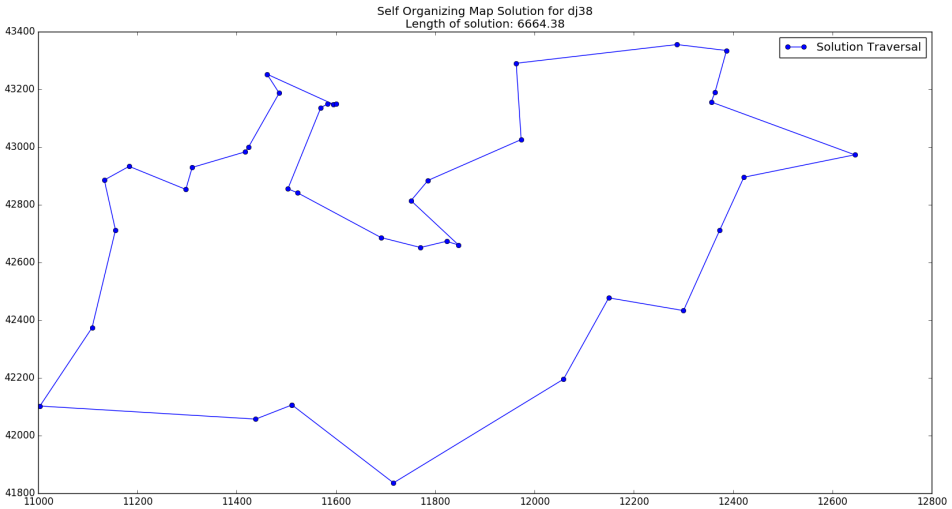

Self Organizing Map Solution for dj38
Length of solution: 6688.92
Solution Traversal
43400
43200
43000
42800
42600
42400
42200
42000
41800
11000
11200
11400
11600
11800
12000
12200
12400
12600
12800

Best result using exponential decay function:





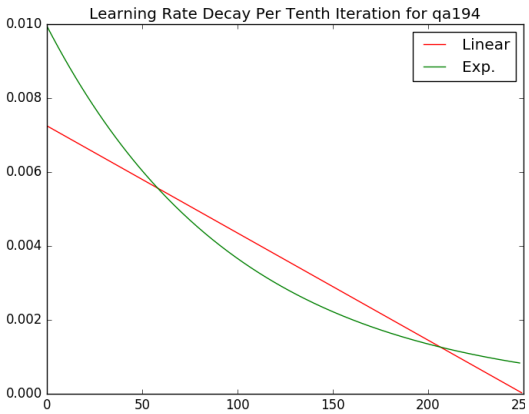

Self Organizing Map #2500
Length of current solution: 6664.38
City Coordinates
Neuron Scaffold
1.00
0.99
0.98
0.97
0.96
0.25
0.26
0.27
0.28
0.29

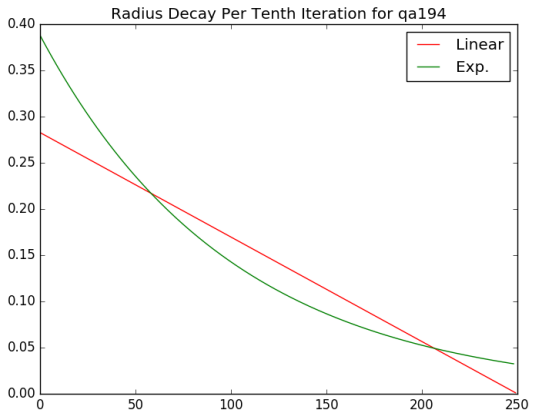

Self Organizing Map Solution for dj38
Length of solution: 6664.38
Solution Traversal
43400
43200
43000
42800
42600
42400
42200
42000
41800
11000
11200
11400
11600
11800
12000
12200
12400
12600
12800

Qatar

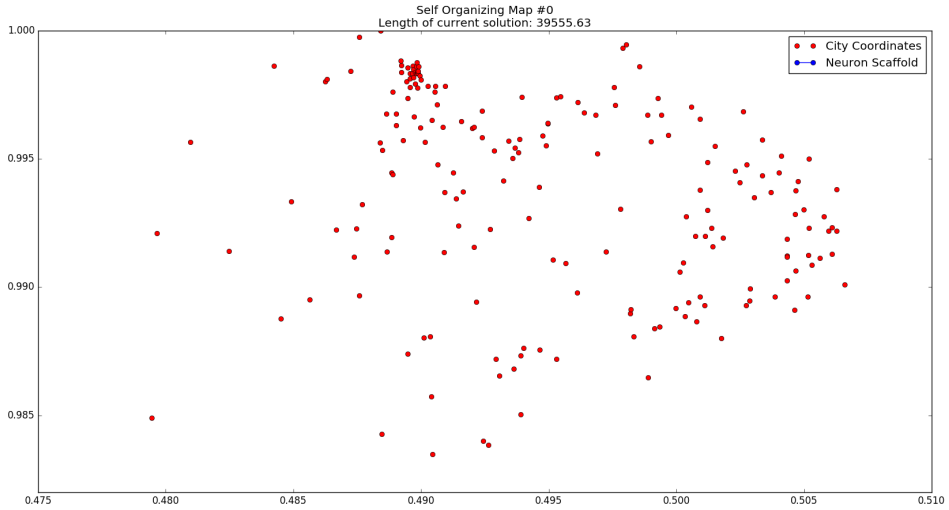
Diagrams for data set qa194 all use the same number of neurons, which is 388. The linear and exponential decay functions have an initial learning rate and initial neighborhood radius of 1 and 39, respectively. The static decay function has learning rate 0.25 and neighborhood radius 20.

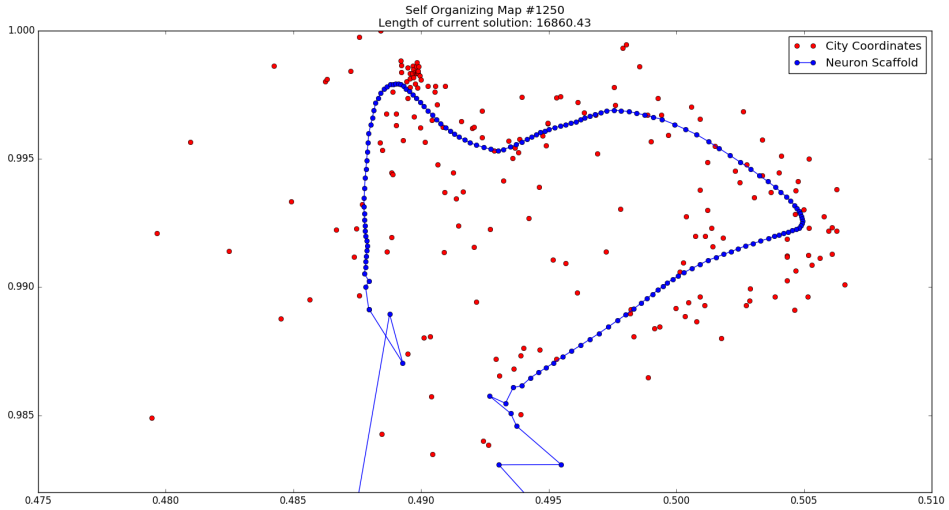
For the static decay function, there will be no change in learning rate or neighborhood size. The learning rate and neighborhood size change for the linear and exponential decay functions are shown in two separate plots:

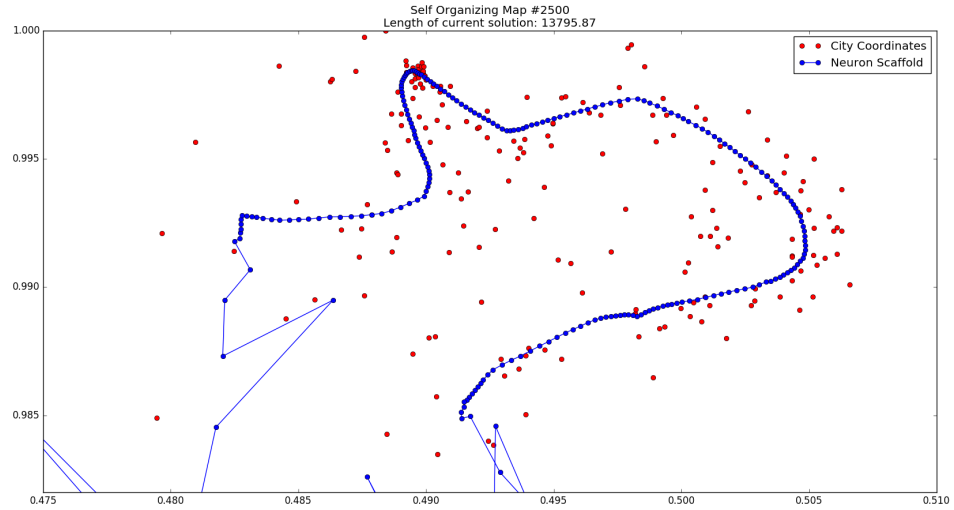


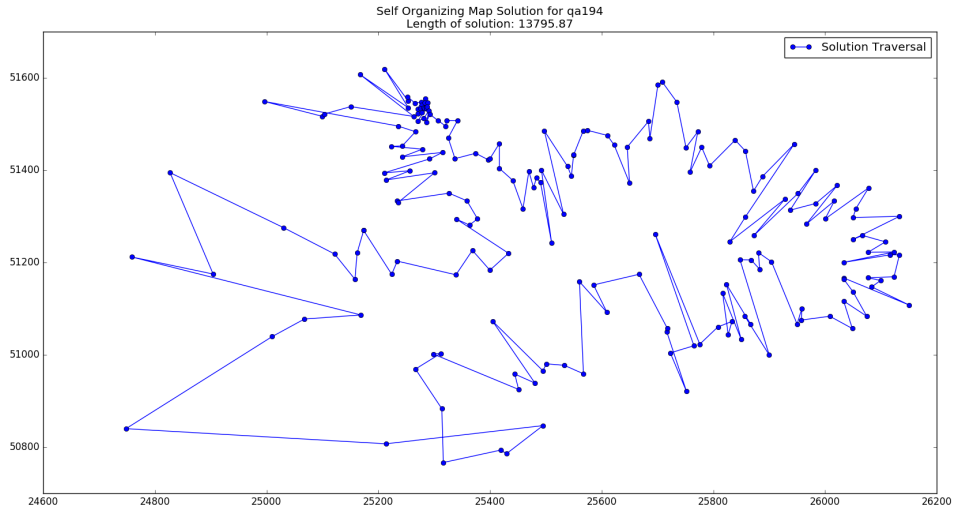


Best result using static decay function:

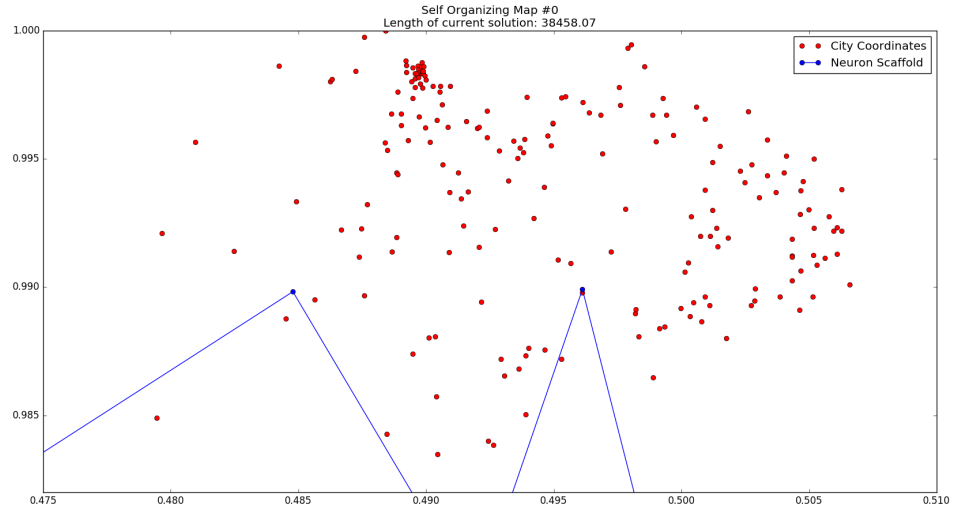


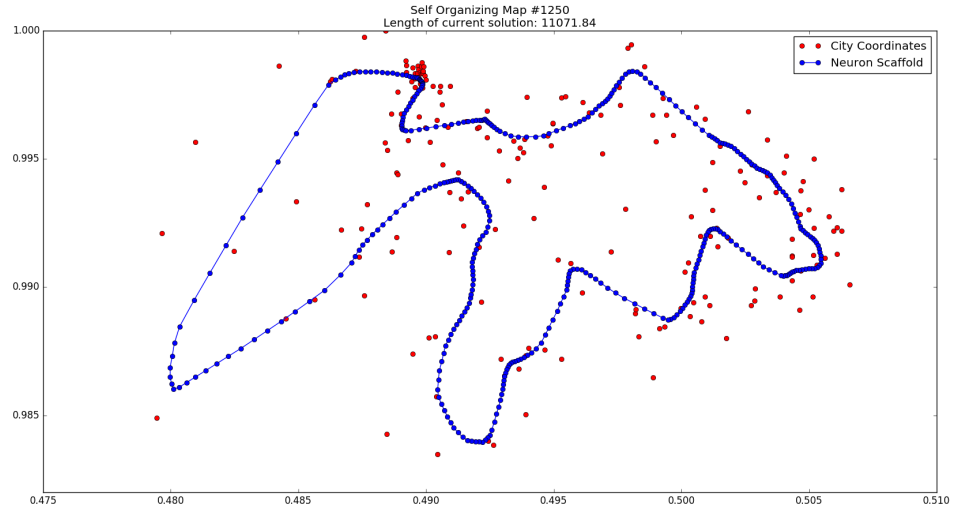


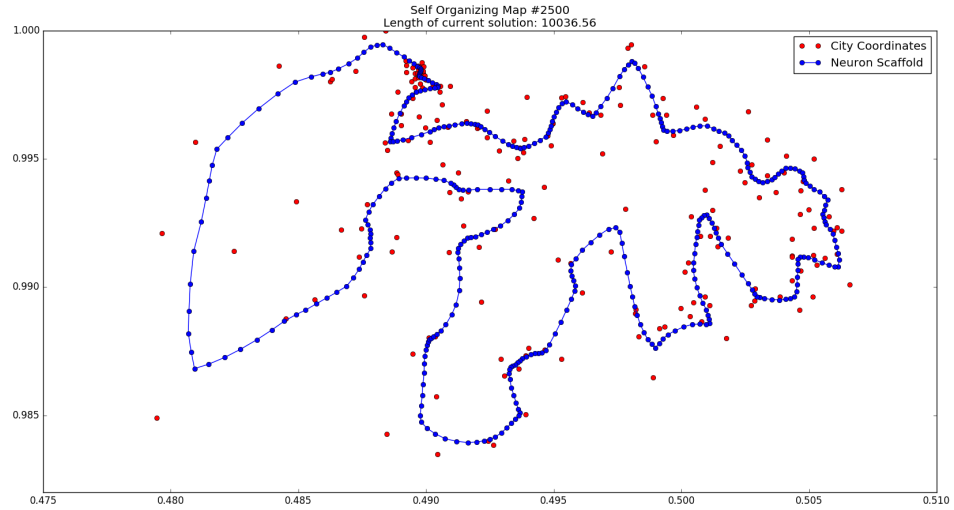

Self Organizing Map #2500
Length of current solution: 13795.87
City Coordinates
Neuron Scaffold
1.000
0.995
0.990
0.985
0.475
0.480
0.485
0.490
0.495
0.500
0.505
0.510

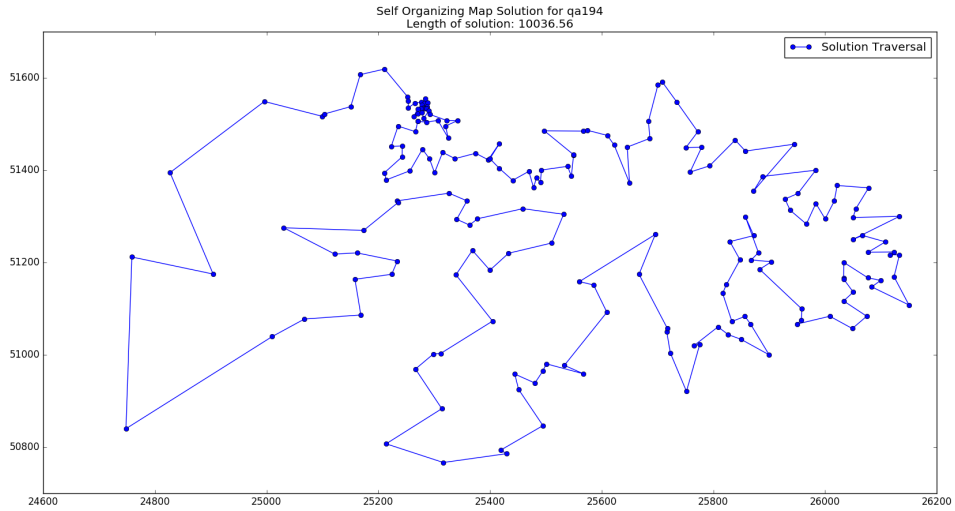

Self Organizing Map Solution for qa194
Length of solution: 13795.87
Solution Traversal
51600
51400
51200
51000
50800
24600
24800
25000
25200
25400
25600
25800
26000
26200

Best result using linear decay function:

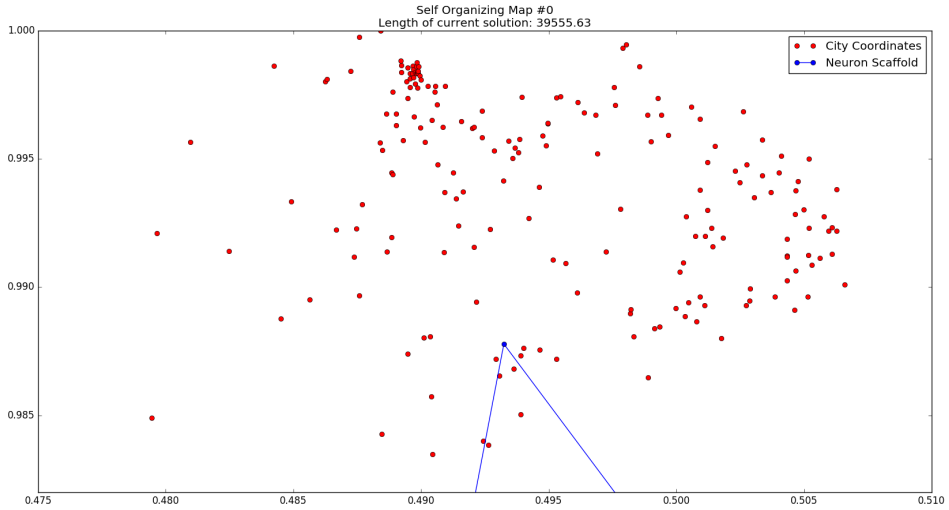


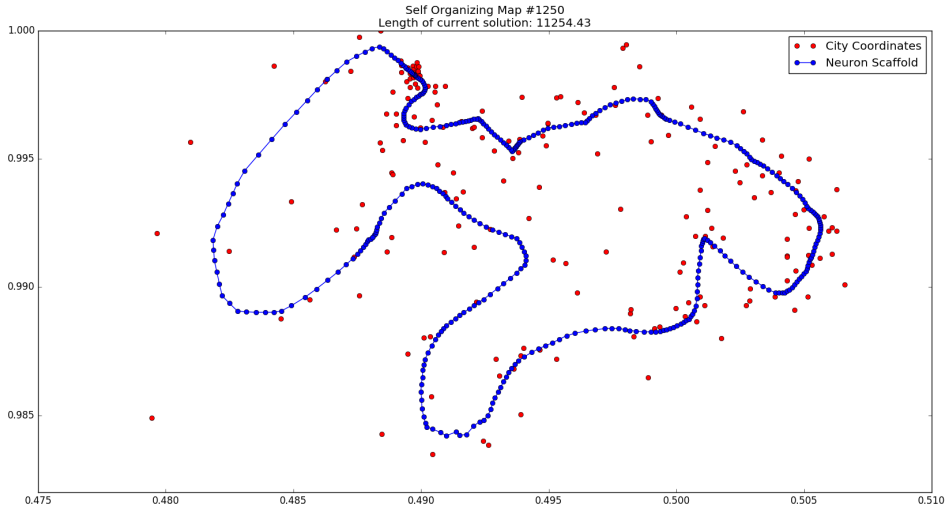


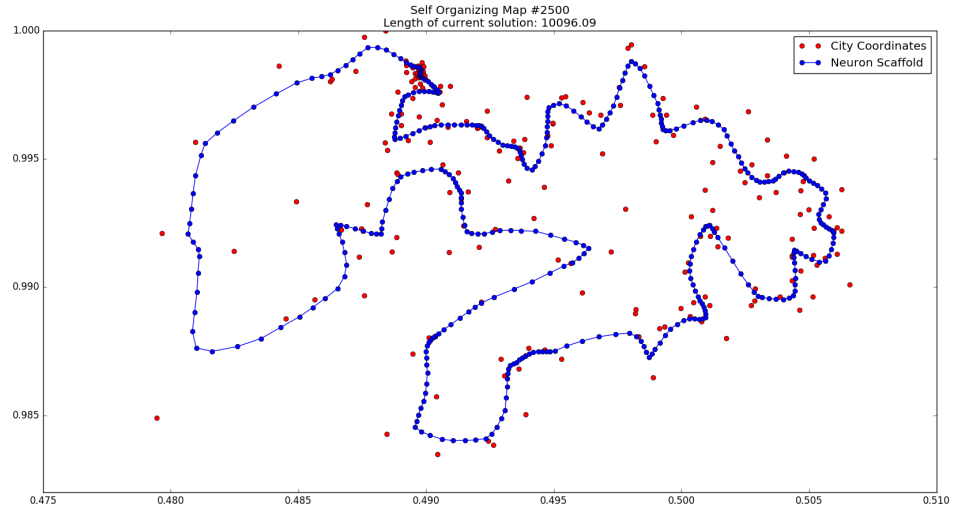

Self Organizing Map #2500
Length of current solution: 10036.56
City Coordinates
Neuron Scaffold
1.000
0.995
0.990
0.985
0.475
0.480
0.485
0.490
0.495
0.500
0.505
0.510

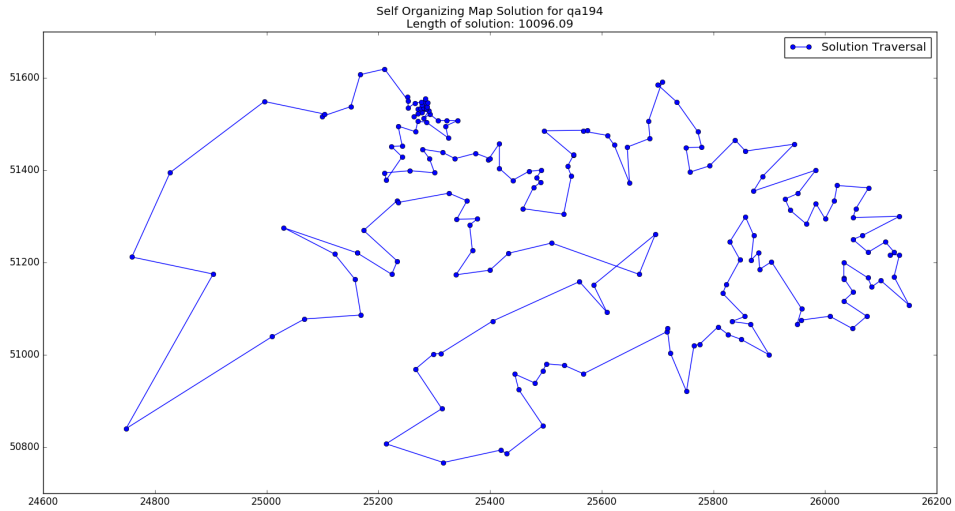

Self Organizing Map Solution for qa194
Length of solution: 10036.56
Solution Traversal
51600
51400
51200
51000
50800
24600
24800
25000
25200
25400
25600
25800
26000
26200

Best result using exponential decay function:





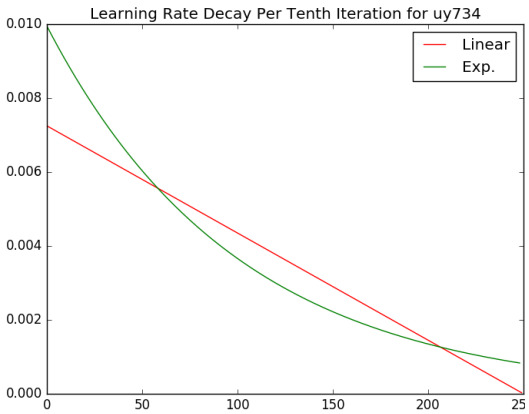

Self Organizing Map #2500
Length of current solution: 10096.09
City Coordinates
Neuron Scaffold
1.000
0.995
0.990
0.985
0.475
0.480
0.485
0.490
0.495
0.500
0.505
0.510

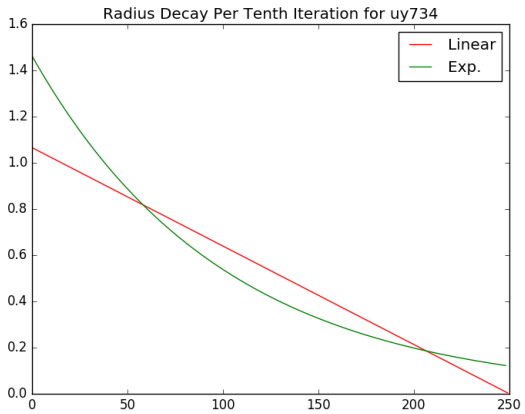

Self Organizing Map Solution for qa194
Length of solution: 10096.09
Solution Traversal
51600
51400
51200
51000
50800
24600
24800
25000
25200
25400
25600
25800
26000
26200

Uruguay

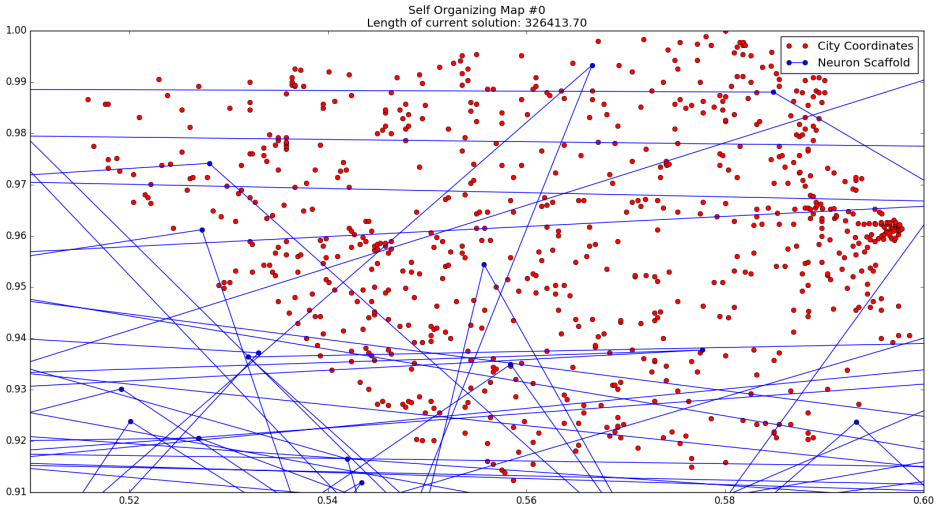
Diagrams for data set uy734 all use the same number of neurons, which is 1468. The linear and exponential decay functions have an initial learning rate and initial neighborhood radius of 1 and 147, respectively. The static decay function has learning rate 0.25 and neighborhood radius 74.

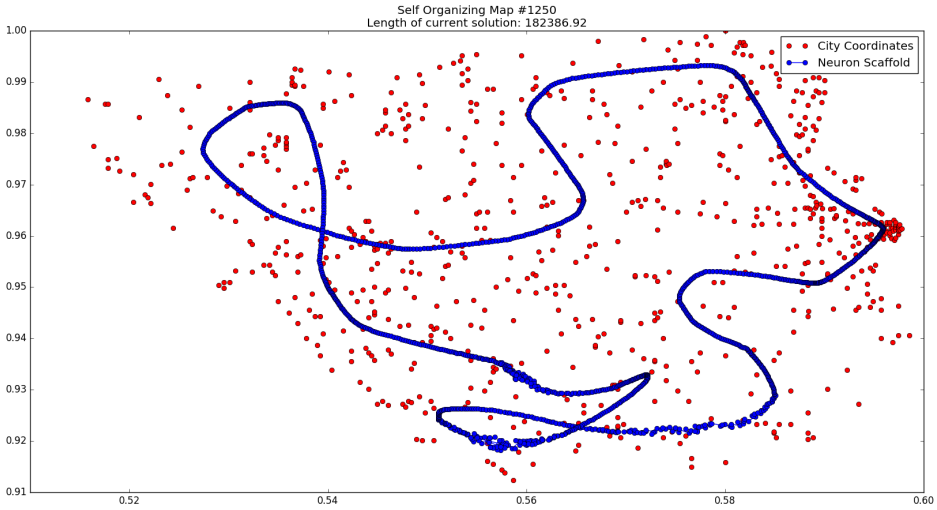
For the static decay function, there will be no change in learning rate or neighborhood size. The learning rate and neighborhood size change for the linear and exponential decay functions are shown in two separate plots:

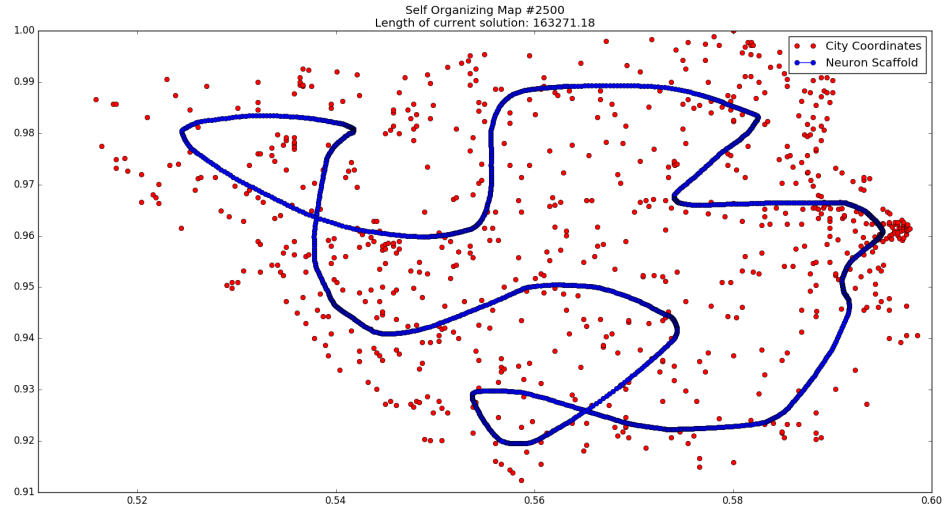


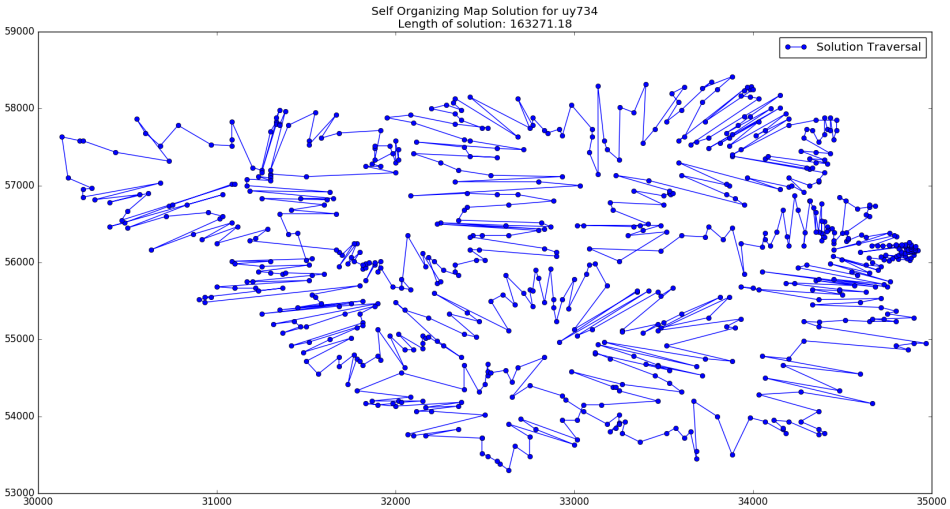


Best result using static decay function:

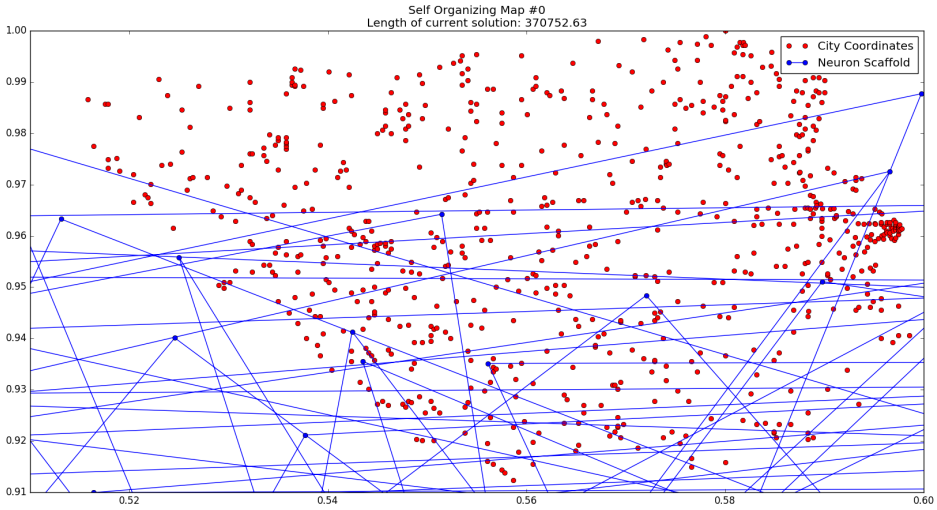


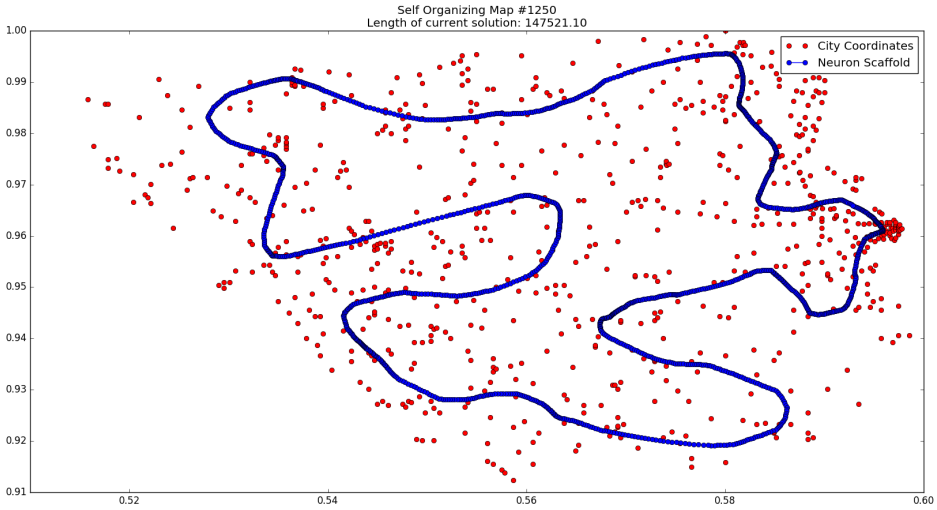


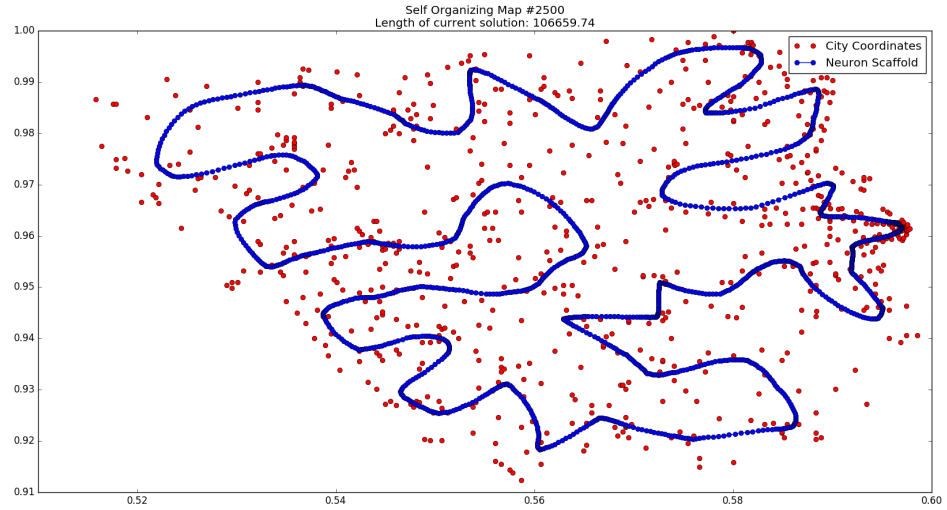

Self Organizing Map #2500
Length of current solution: 163271.18
City Coordinates
Neuron Scaffold
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.52
0.54
0.56
0.58
0.60

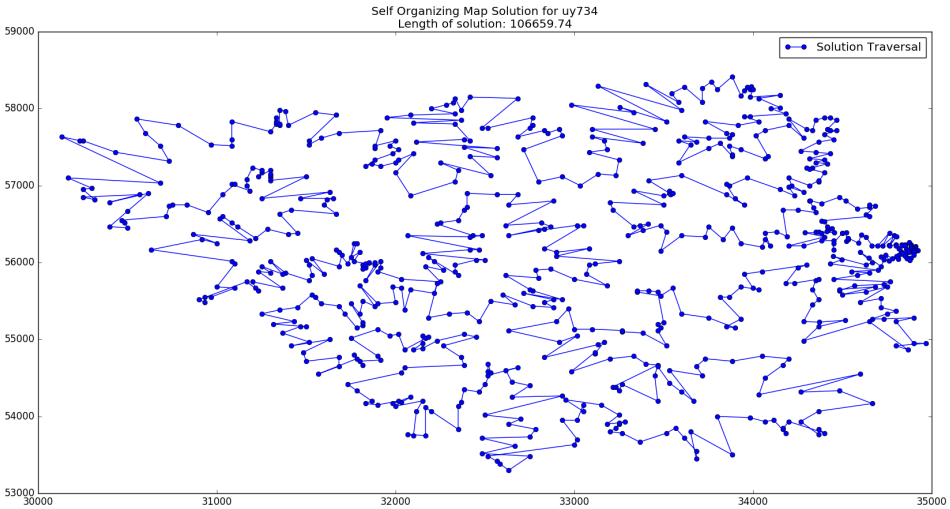

Self Organizing Map Solution for uy734
Length of solution: 163271.18
Solution Traversal
59000
58000
57000
56000
55000
54000
53000
30000
31000
32000
33000
34000
35000

Best result using linear decay function:

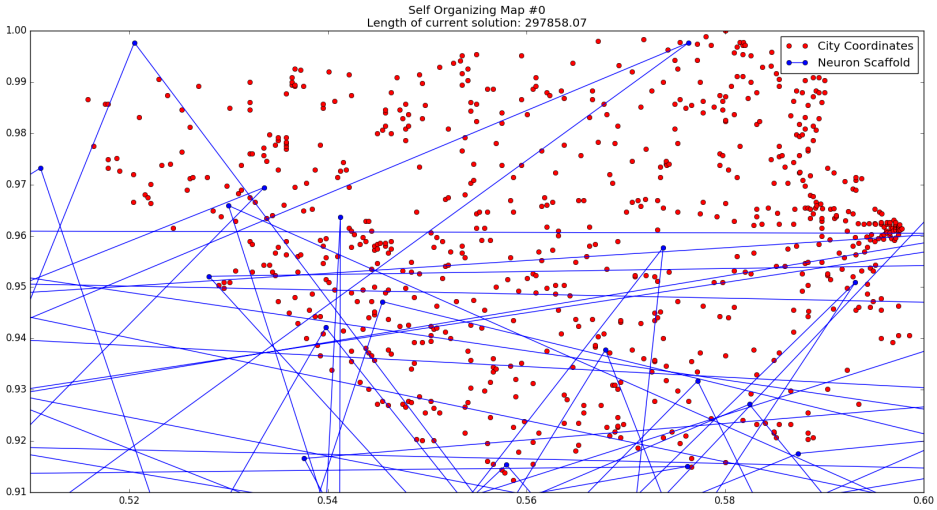


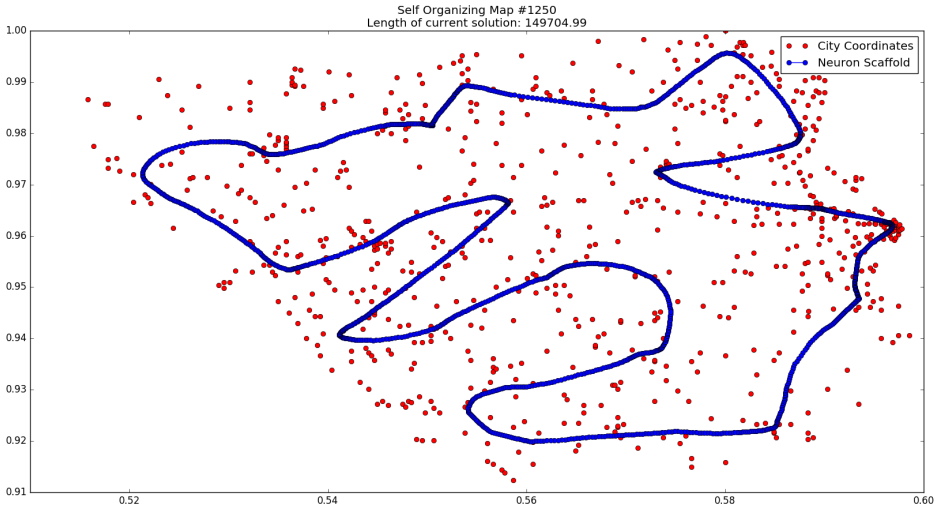


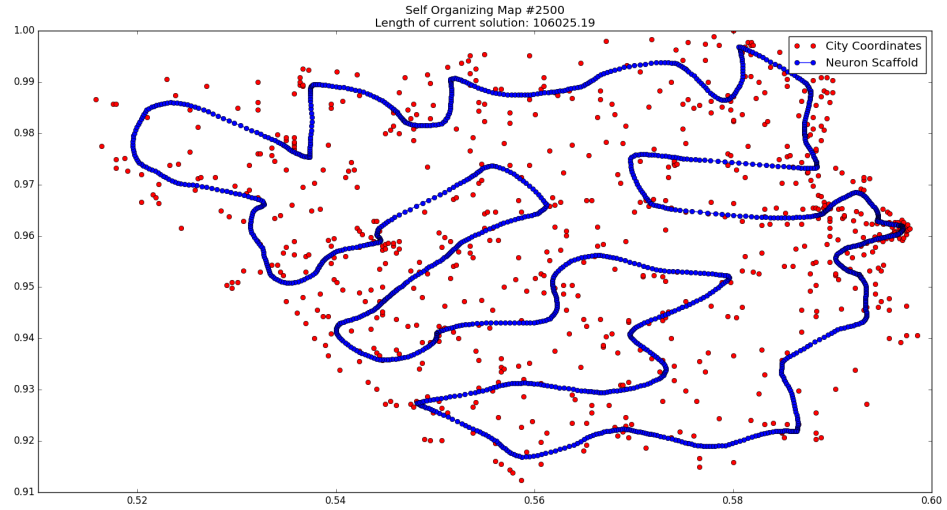

Self Organizing Map #2500
Length of current solution: 106659.74
City Coordinates
Neuron Scaffold
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.52
0.54
0.56
0.58
0.60

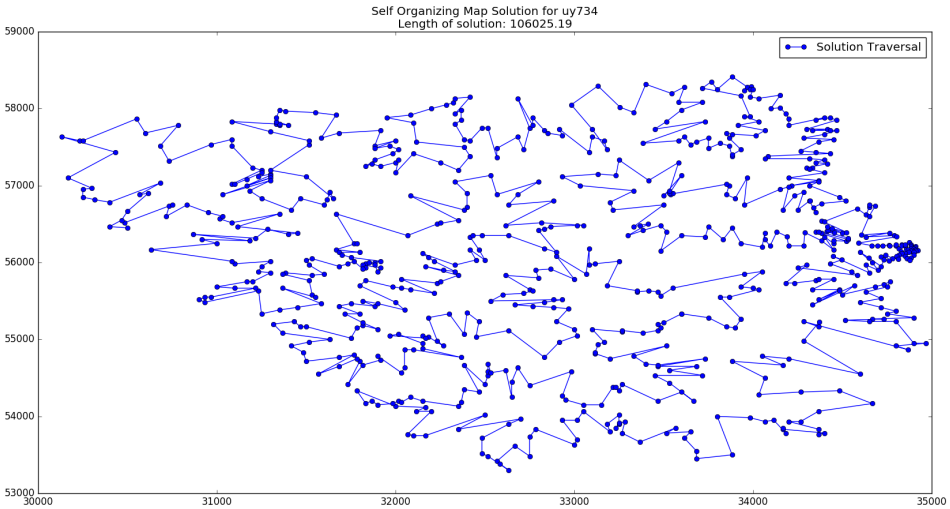

Self Organizing Map Solution for uy734
Length of solution: 106659.74
Solution Traversal
59000
58000
57000
56000
55000
54000
53000
30000
31000
32000
33000
34000
35000

Best result using exponential decay function:






Self Organizing Map #2500
Length of current solution: 106025.19
City Coordinates
Neuron Scaffold
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.52
0.54
0.56
0.58
0.60


Self Organizing Map Solution for uy734
Length of solution: 106025.19
Solution Traversal
59000
58000
57000
56000
55000
54000
53000
30000
31000
32000
33000
34000
35000