

Homework 3 -- due Friday 26.1.2018, 23:59:59

Problem 1: [Vector Quantization](#) (30p)

In this problem you should group 2d input points (x,y) into clusters and determine the center of each cluster.

The number of required clusters is provided as integer number on the first line.

Following, the system provides an unknown number of 2d input data points (x, y), one per line.

Continue reading

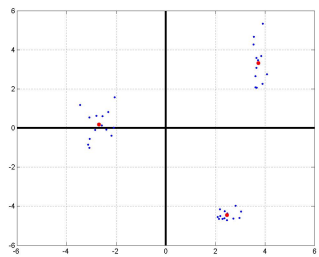
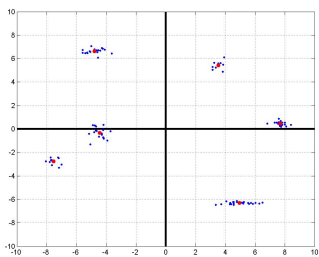
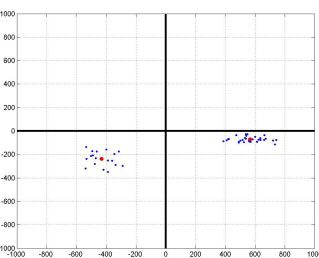
until your program obtains no more data. You can safely assume to read less than 1000 points. After reading, you

should run the Vector Quantization algorithm to find the center(s) of input data, and finally report the center position

as x, y coordinate. Present one such center position per output line. The order of center points output does not

matter. We will accept deviations of center positions within 2% of the overall input range (so e.g. if you find input

from -10 .. +10 (total 20), we will accept +/- 0.4 offset).

Example Input	Visualization	Required Output
testInput21A.txt	 <p>"click" for a large view blue dots: input data red dots: centers</p>	testOutput21A.txt
testInput21B.txt	 <p>"click" for a large view blue dots: input data red dots: centers</p>	testOutput21B.txt
testInput21C.txt	 <p>"click" for a large view blue dots: input data red dots: centers</p>	testOutput21C.txt

http://ci.nst.ei.tum.de/ci_ws2017/homework/hw3/hw3.html

```

. . . * * * * . . . * * * . * * . . . .
. . . . . * * * * * * . * * * * . . . .
. . . . . . . . . . . . . . . . . . . .
-
. . . . * * * * * . . . . * * * * * * * .
. . . * * * * . * . * * * * * * * * .
. . * * * . * * * * * * * * * * . * .
. . * * * * * * * * * * * * * * * * .
. . . . * * * * * . . . . * * * . .
. . . . * * * * . . . . * * . * . . .
. . . . * * * * * . * . . . . . . . .
. . * . . . . * . * * . . . . . . . .
. . * . . . . * * * . . . . * * . .
. . * . . . . * . . . . . . . . . .

```

Problem 3: [Self Organizing Maps](#) (35p)

Your program shall find a solution path for the Traveling Salesman Problem (finding a short path to travel once to each city and return home), for an unknown number of cities as input (you can safely assume ≤ 1000 cities). Each city consists of an ID (an integer number), and X and Y position of that city (two integer numbers). The provided input format for each line to read in is

CITY-ID, X, Y\n

Your program shall implement a Self-Organizing Map to accomplish this task. When your SOM finished learning, print the path as one city-id per line, followed by '\n'. Example for three cities with IDs 1,2,3 which are visited in the order 3,1,2:

```

3\n
1\n
2\n

```

Remember that the number of cities in the output corresponds exactly to the number of cities in the input. It does not matter which of the cities is the first on your path.

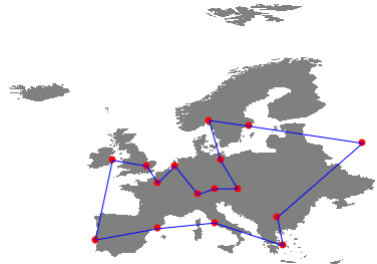
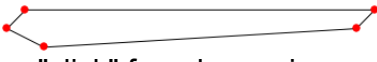
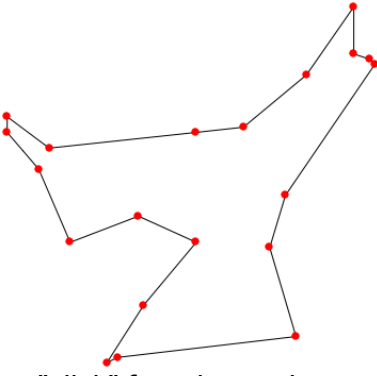
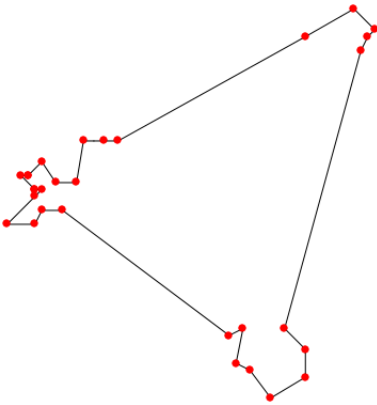
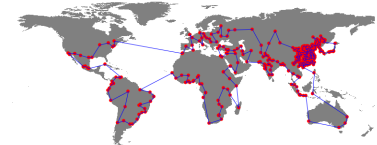
You can safely assume that your program does not need to find the shortest possible path (remember, this problem is NP hard!), but your result needs to be within 15% of the shortest path we found (which again might not be optimal).

Hints, tricks, and things to keep in mind:

-) Once your SOM has "relaxed", it is well possible that some cities are not covered by a neuron whereas several neurons might have specialized on the same city. Therefore, compute your path as follows: For each city, select the nearest neuron and remember this association. Afterwards, iterate over all neurons in order and emit all cities which are connected to this neuron. The path (order) amongst multiple city associated to the same neuron is irrelevant, as this adds only small distances for travel compared to the large inter-neuron distances.
-) Remember to normalize your inputs; or initialize your SOM neurons according to the input space to cover.
-) Ensure that you don't accidentally divide by zero (which might well happen if the neighborhood-function weakens during adaptation)
-) The population size does not need to correspond to the exact number of cities. In fact, the results are typically better if the population size is slightly bigger than the number of cities.
-) As the SOM often does not finish learning within the granted timewindow (here 5 minutes == 300 seconds per data set), use the 'time()' function to keep track of the time your application runs, possibly stopping briefly before timeout, and display the intermediate result as final solution. Most likely your "intermediate" solution will be good enough for us. Here is an example for the time() function:

```
#include <time.h>
```

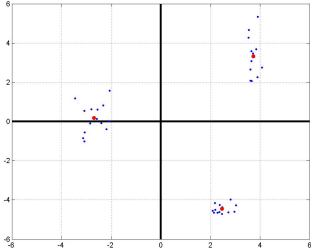
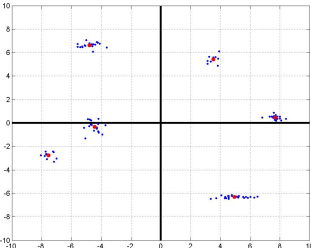
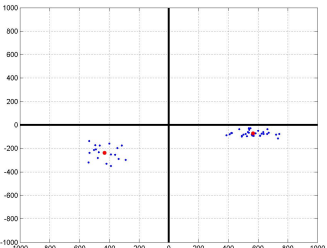
```
time_t t0 = time();
// your code heres
time_t t1 = time();
printf("The application took %lus\n", t1 - t0);
```

Example Input	Visualization	Required Output (one possible solution)
testInput23A.txt	 <p>"click" for a large view red dots: input data blue line: computed path</p>	testOutput23A.txt
testInput23B.txt	 <p>"click" for a large view red dots: input data blue line: computed path</p>	testOutput23B.txt
testInput23C.txt	 <p>"click" for a large view red dots: input data blue line: computed path</p>	testOutput23C.txt
testInput23D.txt	 <p>"click" for a large view red dots: input data blue line: computed path</p>	testOutput23D.txt
testInput23E.txt	 <p>"click" for a large view</p>	testOutput23E.txt

red dots: input data
blue line: computed path

Problem 4: [Vector Quantization \(BONUS\)](#) (20p)

Exact same setting as in Problem 1, but here the number of clusters is unknown. So you will not read in the number of clusters in the first line, but instead start with reading in data points right away. It is your program's task to determine the number of clusters in addition to the center position. Do not print the number of clusters that you found (as this is implicitly given by the number of output lines).

Example Input	Visualization	Required Output
testInput24A.txt	 <p>"click" for a large view blue dots: input data red dots: centers</p>	testOutput24A.txt
testInput24B.txt	 <p>"click" for a large view blue dots: input data red dots: centers</p>	testOutput24B.txt
testInput24C.txt	 <p>"click" for a large view blue dots: input data red dots: centers</p>	testOutput24C.txt

You are done!