Project 2

Consider the network topology design problem described below.

Given the location of n nodes in the plane by their coordinates. Create a network topology (represented by an undirected graph), such that it has the following properties:

1. It contains all the given nodes.
2. The degree of each vertex in the graph is at least 3, that is, each node is connected to at least 3 other nodes.
3. The diameter of the graph is at most 4. By this we mean that any node can be reached from any other node in at most 4 hops. (This can be checked by any shortest path algorithm, or by running a breadth-first search from every node.) That is, the diameter in our case refers to the hop-distance, not to any kind of geometric distance. Note that this diameter bound implies that the graph must be connected.
4. The total cost of the network topology is as low as possible, where the cost is measured by the total *geometric length* of all links. This means, you have compute how long each link is *geometrically* (that is, how far apart are its end-nodes), and then sum it up for all links that exist in the network. This sum represents the total cost that we would like to minimize, under the constraints described above in items 1,2,3. Note: do not confuse the geometric distance with the hop-distance, used in the previous point.

The goal is to create and implement **two different heuristic algorithms** for this network topology design problem, and experiment with them. By *heuristic algorithm* we mean that it does not have to guarantee the exact optimum, just a reasonable solution.

The **algorithms** are of your choice. For example, you may use any of the general purpose heuristic optimization algorithms from the course material, including, but not limited to, Branch and Bound, Simulated Annealing, Greedy Local Search, Tabu Search, Genetic Algorithm etc. Or, you can use any other algorithm that you may find by searching the literature, or you may create your own heuristic algorithms. *Creative ideas will be appreciated!* The two

chosen algorithms should be sufficiently different, so that it makes sense to let them compete in finding a good solution for this problem.

Note: It is often the case in practice that one only runs some simple heuristic, adjusted to the specific problem at hand, so it is not necessary to think about a complicated algorithm. As an example, one can start from some random initial solution, and then improve it step by step. The improvement is often just some simple change. For instance, we may delete a random edge from the graph, and add another random edge. If it results in actual improvement, then we accept it, and then repeat from the new situation. If it does not result in improvement, then it is rejected, and we may try again from the previous graph. Eventually, if for a certain time no improvement is found, then we stop, and return the result as a heuristic solution. Many (though not all!) heuristic algorithms are essentially just variants of the above simple principle. You will have the opportunity to play with it, and try your own creative ideas.

Tasks:

1. Describe the two algorithms you want to use for the problem. If something is from the literature, or from the Internet, provide reference to the source. Describe how both algorithms work. First briefly explain informally the ideas. Then provide pseudo code for the description of both algorithms, with sufficient comments to make them readable and understandable by a human.

   Note: If you want to use a standard heuristic algorithm, such as Simulated Annealing, Tabu Search, Genetic Algorithm etc. and something is not clear or not fully specified in the general description, do not get stuck! It is a typical real life situation that some details are not clear or not precise in a verbal description. In this case, take into account that in a heuristic algorithm there is no unique "standard" version for a specific application. In particular, the description of a general method does not detail how to apply it to a specific problem. *Finding it out is part of your task!* Therefore, do not hesitate to fill in the missing details using your own creativity.

2. Write a computer program that implements the algorithms. You may use any programming language under any operating system, this is entirely of your choice.

3. Run the program on randomly generated examples (at least 5 examples), with both algorithms. The instances are created as follows. Pick n random points in the plane. This can be done by generating random numbers in some range and taking them as coordinates of the points. The value of n should be chosen such that the problem is not trivially small (say, you have $n \geq 15$), but the running time is still reasonable (it does not run for days). Show how the random input is generated and present the actual data.

4. Represent the results of the experimental runs graphically. For each run (or for a subset of them, to limit the document size), show in a figure the positions of the nodes and the resulting network topology, that is, the graph. The details of the actual graphical representation are left to your choice, but the essential thing is that by looking at the figure the reader can be convinced that the algorithms find reasonable solutions. It is important that the figures need to be generated by software. If you compute only numerical values and then draw "by hand" from these numbers, that is not a professional solution and will result in reduced score.

   Remark: It might be helpful to think about the whole project that your task is not only to solve the technical problem, but you also have to "sell" the results. Try to look at your work from the viewpoint of a potential "customer": how convincing your presentation would look for a customer? If you were in the customer's shoes, would you buy the software?

5. Draw some conclusion about how the two algorithms compare. Is one of them always better? If so, how much better? If the algorithms are iterative, then how does the cost change over the iterations? How many iterations are needed to reach a solution that does not improve significantly in further iterations? How does the experimental running times in the algorithms depend on the size of the problem? (You may run more experiments, if needed. The examples mentioned in item 3 represent only a minimum set of experiments.)

Formatting and submission guidelines: same as for Project 1.