Neural and Evolutionary Computation (NEC)

A4: Optimization with Genetic Algorithms

Objective

Implementation of a genetic algorithm (GA) for the clustering of the nodes of a graph, by means of the optimization of modularity.

Modularity

Let G be an undirected network (graph) with N vertices, L edges, no self-loops and adjacency matrix a_{ij}. If we have a partition C of the nodes of the graph in disjoint clusters, the *modularity* Q is defined as

$$Q(C) = \frac{1}{2L}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(a_{ij} - \frac{k_i k_j}{2L}\right)\delta\left(C_i, C_j\right)$$

where k_i is the degree (number of edges) of node i, and $\delta(C_i,C_j)$ is 1 if nodes i and j belong to the same cluster, 0 otherwise. More precisely,

$$k_i = \sum_{j=1}^{N} a_{ij}$$

$$2L = \sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij} = \sum_{i=1}^{N} k_i$$

Modularity is zero when all nodes belong to the same cluster, and in general it may take values in the range

$$-1 \le Q(C) \le 1$$

The modularity of a given partition is then the probability of having edges falling within groups in the network minus the expected probability in an equivalent (null case) network. This null case network has the same number of nodes as the original network and the edges placed at random preserving the nodes' degrees. Having a large modularity value means a higher deviation from the null case, and therefore a better partitioning of the network. Note that the optimization of the modularity cannot be performed by exhaustive search since the number of different partitions is equal to the Bell or exponential numbers, which grow at least exponentially in the number of nodes. Indeed, optimization of modularity is a NP-hard problem.

We are only interested in partitions of the nodes in two clusters, which we will call the *left* and *right* clusters respectively. Therefore, if we define S_i as 0 if node i belongs to the left cluster, and 1 if it belongs to the right cluster, then the partition in two clusters is completely determined by the vector S, and

$$\delta\left(C_i, C_j\right) = S_i S_j + \left(1 - S_i\right)\left(1 - S_j\right)$$

Genetic algorithm

Given the network *G*, any vector *S* may be seen as the chromosome corresponding to a partition in two clusters, and the fitness is related to its modularity *Q(S)*. The objective is the implementation of a genetic algorithm to obtain the partition which maximizes modularity.

Fitness

Modularity cannot be used directly as the fitness since it may take negative values, and also the difference in modularity of good partitions may be very small. You should define a transformation of modularity into an adequate positive fitness function, use the rank, or try with robust selection methods.

Data

- **Input:** a network in *Pajek* format (*.net) (http://pajek.imfm.si/). Note that, for undirected networks, $a_{ij}=a_{ji}$, despite the file may contain just one of the pairs (*i*, *j*) or (*j*, *i*). That is, ensure the adjacency matrix is symmetric
- **Output:** the highest modularity partition found, in *Pajek* format (*.clu), and its corresponding value of modularity

Parameters

Try different variants of the genetic algorithm and of their corresponding parameters, for instance:

- Fitness function
- Selection scheme: proportional (roulette-wheel), truncation, ranking, tournament, fitness uniform (FUSS), Boltzmann, etc.
- Crossover: one-point crossover, two-point crossover, uniform crossover, etc.
- Mutation
- Elitism

Guidelines

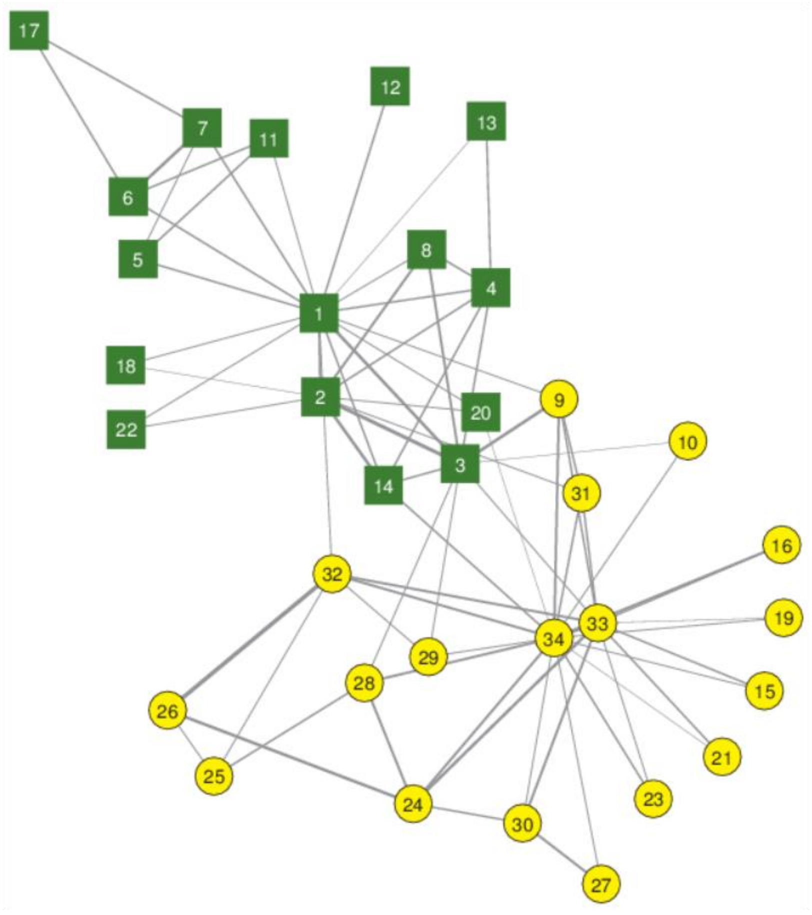
- Individual practical exercise, not in group
- You must try to optimize modularity for “all” the provided networks
- GA can be implemented in any language, but compiled languages (e.g. C, C++, Julia, C#, Java,, Ada) are preferred due to its much faster execution time. Note that GA training may take many hours when using interpreted languages (e.g. Python, Matlab, Octave, R)
- Check if your program is calculating the modularity properly, using e.g.:
    - in Pajek, loading the network and the partition, and going to Operations → Network+Partition → Info → Modularity
    - in Radatools, using the *Modularity_Calculation* program
    - in LightGraphs (Julia), Networkx (python), or igraph (R, python, C++), using their respective *modularity* functions
- Use of libraries which already implement GA is forbidden

Delivery

- The delivery must be done in a compressed file (e.g. zip, rar) whose name should be of the form:
    - A4_NameSurname.zip
- The delivery must contain:
    - Source code of GA
    - Input files, if any (e.g. configuration file with the GA parameters)
    - Result files (best partitions) for all the provided networks
    - Report in pdf:
        - Description of the implementation (language, tools used, etc.)
        - Implementation decisions
        - Results: plot (graph + partition) and modularity value of the best partition found for all the provided networks

Example

This is the "real" partition of the Zachary Karate Club network, as a reference for a good partition in two communities (not necessarily the one with largest modularity). Its modularity is Q = 0.37146614.


17
12
13
7
11
6
5
8
4
1
18
2
20
9
22
10
3
14
31
16
32
33
19
34
29
28
15
26
21
25
23
24
30
27