1. A genetic algorithm to cluster graphs [70 points]

Finding dense modules or clusters in a graph is an important part of many data mining problems. One popular definition of a 'module' is a set of nodes that have many more within-module connections (i.e. connections between nodes in the same module) than between-module connections (i.e. connections between nodes in different modules) than expected by chance. In 2002, Newman proposed an objective function, called *modularity*, that characterizes the quality of a clustering C of a graph $G = (V, E)$:

$$q(G, C) := \sum_{u,v \in V} (A_{uv} - k_u k_v/(2m))(1 - x_{uv}), \tag{1}$$

where A_{uv} is 1 if u and v have an edge in E and 0 otherwise; k_u is the degree of node u (i.e. its number of neighbors); m is the total number of edges in the graph; and the variables x_{uv} describe C by indicating which nodes are in the same module. Specifically, for every pair of nodes, $x_{uv} = 0$ if u and v belong to the same module, and $x_{uv} = 1$ otherwise. Notice that there is no contribution towards the modularity score for a pair of nodes that lie in different modules and that all terms (A_{uv}, k_u, k_v, m) are fixed besides the x_{uv} terms.

The goal is to find the clustering C that maximizes this function. In general, the clustering C can have any number of modules (from 1 to n, where n is the number of nodes in the graph), but all nodes must be assigned to exactly one module.

Q1.1 Write a genetic program to cluster an input graph into modules that optimizes the Newman objective function *using at most 5 clusters.*

We have provided an example network for you to test your genetic algorithm (`karate.txt`, downloadable from the class website). The first line of this file begins with a hash `'#'` followed by the number of nodes and edges in the following format:

```
#n=34,m=78
```

Each following line is **tab-delimited** and contains the two end-points of each edge:

```
1 0
2 0
...
33 32
```

In general, if there are m edges in the graph, there should be exactly $m + 1$ lines in the file. All graphs are undirected and edges are repeated only once (e.g. in the above example, there will not be a second line with `'0 1'` since this is the same as the first line for undirected graphs).

Expected output: Your program should output the optimal modularity found as well as the clustering corresponding to the optimal modularity. Specifically, the first line of the output should contain the modularity score in the following format:

```
#modularity=0.419790
```

The following lines should contain the clustering with one line per module and specifying which nodes were assigned to that module:

```
Module 1:  0 1 2 3 6
Module 2:  4 5 10
...
Module k:  33 9 11
```

What you should submit: an **executable** program, written in any language you want, that takes one **command-line** parameter corresponding to a filename which contains the graph data (e.g. `karate.txt`). It is very important that you can take input from any graph in the described format because we will test your algorithm on different graphs. You should also include a `README.txt` file explaining how to compile and run your program on the example network. If your program does not run, you will get 0 points.

Submission: Please email a zip file containing your code and a `README.txt` to `navlakha@cs.cmu.edu` before the deadline.

In addition, please answer the following questions about your design.

Q1.2 For this graph clustering problem, what is the analog of an 'individual' in the sense that it was presented in lecture?

A clustering.

Q1.3 What is the 'fitness' function?

The modularity function.

Q1.4 Describw how you represented or encoded a solution (i.e. a clustering C) in your genetic algorithm?

Q1.4 What were the main ideas behind your solution? Which operations did you use (mutation, cross-over, etc.), and how did you ensure that they always produced a valid clustering (e.g. how did you ensure that a node was not assigned to multiple clusters?)?

2. Short-answer questions about genetic algorithms [20 points]

2.1 In genetic algorithms, a *schema* is a template that defines a set of possible strings. For example, the schema $H = (0\ *\ *\ 1\ *\ 1\ *\ *\ 0\ *)$ implies that the first bit of the string must be 0, but the second bit can either be 0 or 1, etc. A string *destroys* the schema if the string is not a possible string defined by the schema. What is the probability that H will not be destroyed by mutation that occurs with probability 1/12?

The prob that positions 1, 4, 6, and 9 aren't mutated is: $(1-\frac{1}{12})^4$

[5 pts]

2.2 The fitness f of a string of bits s with $l = 4$ is defined as the integer value representation of a (e.g. $f(0101) = 5$). What is the average fitness of the schema $(1***)$ under f? What is the average fitness of the schema $(0***)$ under f?

Avg. fitness of $(1***)$: 11.5

Avg. fitness of $(0***)$: 3.5

[5 pts]

2.3 What is the problem of doing cross-over operations for the traveling salesman problem? What about point mutations? Provide a way to perform the cross-over and mutation operations, respectively, that avoids these issues.

For cross-overs, you could visit the same city twice. Similar issue with point mutations. There are many ways to handle this. E.g. for point mutations: 123456, if a mutation occurs at position 2 (2→4), you could swap with what was previously at 4: 123456 (old). 143256 (new)

[5 pts.]

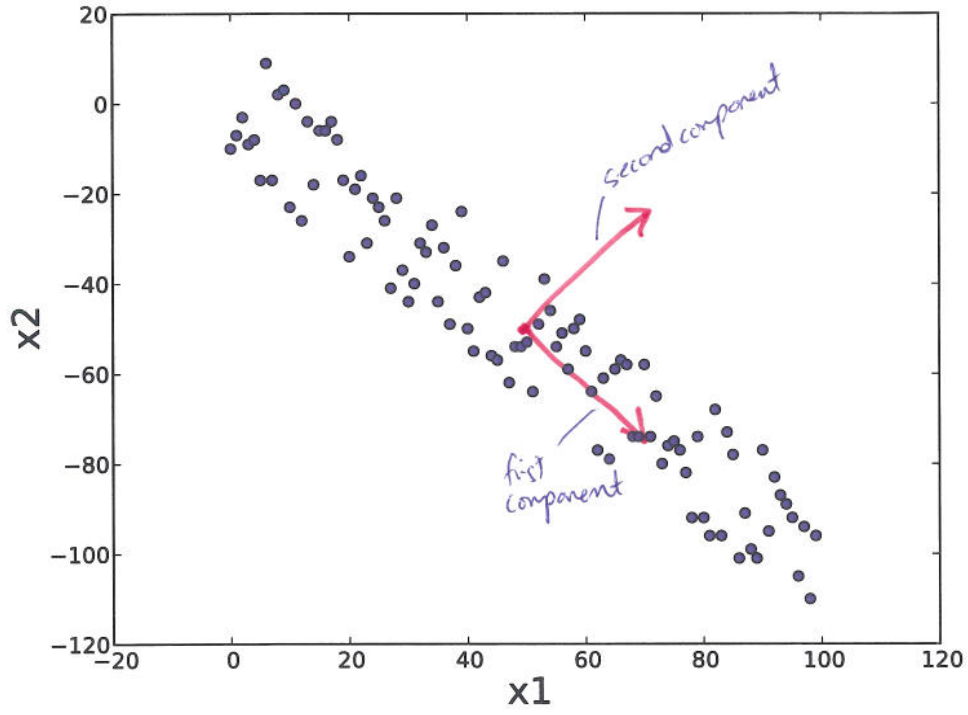
2.4 There are nine possible schemas for a bit string that uses two bits: **00**, **01**, **11**, **11**, **0***, ***0**, **1***, ***1**, ****. How many possible schemas are there for a bit string with l bits?

3^l

[5 pts]

3. Dimensionality Reduction [10 points]

3.1 Given the following data, draw the first two principal components that would be found using the PCA algorithm presented in class.



[6 pts]

3.2 Would the reconstruction error be $> 0, < 0$, or $= 0$ if we were allowed to choose 1 component? What if we could choose 2 components?

1 component: error is > 0.

2 components: error is $= 0$.

[4 pts]