CS-E4600 — Programming project

Part I: Problem description

November 29, 2018

Due: Friday, Dec 21, 2018, midnight

Instructions

In this programming project you have to design and implement your own method to partition a graph into communities. The tasks you have to complete for the programming project are the following:
(1) design and implement your own graph-partitioning method;
(2) submit your solution, which should include the source code you developed and a report; and
(3) use your implementation to participate in a graph-partitioning competition.

To receive full points you only have to complete Tasks (1) and (2). Task (3) is optional, however the top 10 winning teams will receive upto 50 extra points. The deadline to submit your solution (i.e., Task (2)) is Friday, Dec 21, 2018. Detailed instructions for Tasks (2) and (3) will be announced shortly. In this document we describe the problem that you have to solve so that you can start with Task (1).

You can work on the project individually or in two-person teams. You can find your own teammate or ask us to find one for you. To do so please send an email to Suhas (`suhas.muniyappa@aalto.fi`).

Each team needs to return only one solution (that is, we do not require separate solution from each of the team members). Detailed instructions on the content and format of the report will be announced shortly.

The amount of code required for the project is not a lot, however, you may have to think carefully about your algorithm. Do not let the project for the last moment. Start early!

It is also possible to work on a different programming project than the one proposed here. In this case, you should propose your own project. Your project needs to be related to the topic of the course, *algorithmic methods for data analysis*, it needs to be of equivalent difficulty to the one described here, it should not be already implemented, and it should not be used in another course.

If you wish to propose an alternative project, you should write a project description, send it to the course instructor, Aris Gionis (`aristides.gionis@aalto.fi`), and have it approved. The deadline to propose your own project is Monday, Nov 19.

Remember that there is a budget of 5 late days, which you can use in any way you want for the three homeworks and the project. Weekend days count as late days. Late days are counted individually (so a team has to return the project by the time required so that none of the team members exceed their budget).

Project description

We will use graphs from the Stanford Network Analysis Project (SNAP)

`http://snap.stanford.edu/data/index.html`

in particular, you can consider the following 5 *collaboration networks*:

`ca-AstroPh`, `ca-CondMat`, `ca-GrQc`, `ca-HepPh`, and `ca-HepTh`.

These 5 collaboration networks are not (necessarily) the ones that we will use in the competition. However, you can use them as a common basis.

Update: the suggested graph datasets are not necessarily connected and they contain large number of connected components which might result in trivial solutions, in particular, when the value of k is less than the number of connected components. Henceforth, if a graph dataset is not connected then preprocess the graph and consider only the largest connected component of the graph. We have updated the 5 SNAP collaboration networks to be used for this project (Tasks 1 and 2) and it is available in mycourses (link below). Make sure you download the correct preprocessed version of these datasets.

For the competition we will use the following input format: a problem instance (G, k) will be provided as a text file, where the first line specifies the problem parameters (`# graphID numOfVertices numOfEdges k`) and the rest of the lines specify the graph, one line per edge (`vertex1ID vertex2ID`). The 5 SNAP collaboration networks can be found in this format here:

`https://mycourses.aalto.fi/course/resources.php?id=20597`

You can also experiment with other graphs in SNAP. For example, for developing and debugging your code you may want to start with the smallest available graph. In this project we will assume that all graphs are undirected. You can also experiment with directed graphs in SNAP, but you need to treat them as undirected by ignoring the edge directions. If you use other SNAP networks you should transform them by yourself in the project input format.

Graph-partitioning task: Given an undirected graph $G = (V, E)$ and an integer $k > 1$ we want to partition the set of vertices V into k communities $V_1, \ldots, V_k$, so that $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$. We want the communities $V_1, \ldots, V_k$ to be as much separated from each other as possible. We also want that the communities have roughly equal size. Thus, we will evaluate the goodness of a partition $V_1, \ldots, V_k$ using the objective function

$$\phi(V_1, \ldots, V_k) = \frac{|E(V_1, \ldots, V_k)|}{\min_{1 \leq i \leq k} |V_i|},$$

where $E(V_1, \ldots, V_k)$ is the set of edges of G that is "cut" by the k communities, i.e., $E(V_1, \ldots, V_k) = \{(u, v) \in E \mid u \in V_i \text{ and } v \in V_j \text{ with } i \neq j\}$.

You may want to think why small value of the objective ϕ indicates partitions that have well-separated communities and are well balanced.

You should implement a program that reads a problem instance in the format specified above and produces a partition $V_1, \ldots, V_k$ for which $\phi(V_1, \ldots, V_k)$ is as small as possible.

Implementation: Your program should output a partition $V_1, \ldots, V_k$ as a text file with one line per vertex of the form

```
vertexID clusterID
```

There is no restriction on the programming language — you may use the programming language of your preference.

A basic algorithm that you may want to use in the project is the spectral algorithm discussed in class (slide set 12). This algorithm involves forming the Laplacian of the adjacency matrix of the graph, computing and eigen-decomposition of the Laplacian, and then applying k-means on the vector representation of the vertices provided by the eigenvectors. Notice however, that this algorithm does makes any attempt to ensure a balanced partition. So, you may want to think how to improve it in order to obtain a balanced partition (and thus obtain smaller values of the objective ϕ).

For the final grade, 50 point will be given for implementing the basic spectral algorithm, and 50 for designing your own algorithm to achieve balanced partition.

You can also implement a totally different algorithm, if you choose to do so. In this case, to get full points, you will need to justify your proposed algorithm and explain why your algorithm gives small edge cuts and why it favors a balanced partition.

Graph partitioning is a standard graph-mining task and you can find lots of freely available software that implement different algorithms. Obviously you cannot use those implementations. The point of the project is to implement your own algorithm. However, you can use existing implementations of "standard" data-mining tasks. For example, for implementing the basic spectral algorithm, you can use an existing implementation for computing the eigen-decomposition of a matrix, or an existing implementation of the k-means algorithm.

If you are in doubt whether it is OK to use some specific library or routine, please check with the TAs.