# CS-E4600 Algorithmic Methods of Data Mining Programming Project

Adam Ilyas 725819
adam_ilyas@mymail.sutd.edu.sg

December 24, 2018

## 1 Introduction

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

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 , ca−HepTh

File formate first line:

```
# graphID numOfVertices numOfEdges k
```

Subsequent values:

```
vertex1ID vertex2ID
```

## 2 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 $\cap_{i=1}^{k} V_i = V$
We want our communities $V_1, \ldots, V_k$ to be as much seperate from each other as possible. We also want the communities to have roughly equal size. Thus,

we will evaluate the goodness of a partition $V_1, \ldots, V_k$ by **minimizing** 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:

$$E(V_1, \ldots, V_k) = \{(u, v) \in E \mid u \in V_i \text{ and } v \in V_j \text{ where } i \neq j\}$$

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 the objective function $\phi(V_1, \ldots, V_k)$ is as small as possible

## 2.1 Algorithm 1: Unnormalized spectral clustering

**Input**: graph adjacency matrix $A$, number $k$

1. form diagonal matrix $D$
2. form unormalized Laplacian $L = D - A$
3. compute the first $k$ eigenvectors $u_1, \ldots, u_k$ of $L$ (unnormalized Laplacian)
4. form matrix $U \in \mathtt{R}^{n \times k}$ with columns $u_1, \ldots, u_k$
5. consider the i-th row of $U$ as point $y_i \in \mathtt{R}^k,\ i = 1, \ldots, n$
6. cluster (kmeans) the points $\{y_i\}_{i=1,\ldots,n}$ into clusters $C_1, \ldots, C_k$

**output** clusters $A_1, \ldots A_k$

## 2.2 Algorithm 2: Normalized spectral clustering (generalized eigenproblem)

**Input**: graph adjacency matrix $A$, number $k$

1. form diagonal matrix $D$
2. form unormalized Laplacian $L = DA$
3. compute the first $k$ eigenvectors $u_1, \ldots, u_k$ of the *generalized eigenproblem* $L\mathbf{u} = \lambda D\mathbf{u}$ (eigenvectors of $L_{rw}$)

4. form matrix $U \in \mathtt{R}^{n \times k}$ with columns $u_1, \ldots, u_k$
5. consider the i-th row of $U$ as point $y_i \in \mathtt{R}^k,\ i = 1, \ldots, n$
6. cluster (kmeans) the points $\{y_i\}_{i=1,\ldots,n}$ into clusters $C_1, \ldots, C_k$

**output** clusters $A_1, \ldots A_k$

This algorithm is similar to Algorithm 1. The difference is in
- step 3 where we find the eigenvectors $u_1, \ldots, u_k$ of the *generalized eigenproblem* instead of the *unnormalized laplacian*

$$L_{rw} := I - D^{-1}A$$

## 2.3 Algorithm 3: Normalized spectral clustering (generalized eigenproblem, normalized U)

This algorithm is the same as Algorithm 2 but we *normalize* $U$ so that rows have norm 1
**Input**: graph adjacency matrix $A$, number $k$

1. form diagonal matrix $D$
2. form unormalized Laplacian $L = DA$
3. compute the first $k$ eigenvectors $u_1, \ldots, u_k$ of the *generalized eigenproblem* $L\mathbf{u} = \lambda D\mathbf{u}$ (eigenvectors of $L_{rw}$)
4. form matrix $U \in \mathtt{R}^{n \times k}$ with columns $u_1, \ldots, u_k$
5. *normalize* $U$ so that rows have norm 1
6. consider the i-th row of $U$ as point $y_i \in \mathtt{R}^k,\ i = 1, \ldots, n$
7. cluster (kmeans) the points $\{y_i\}_{i=1,\ldots,n}$ into clusters $C_1, \ldots, C_k$

**output** clusters $A_1, \ldots A_k$

## 2.4 Algorithm 4: Normalized spectral clustering (normalize U)

**Input:** graph adjacency matrix $A$, number $k$

1. form diagonal matrix $D$

2. form normalized Laplacian $L' = I - D^{-1/2}AD^{-1/2}$
3. compute the first $k$ eigenvectors $u_1, \ldots, u_k$ of $L'$
4. form matrix $U \in \mathtt{R}^{n \times k}$ with columns $u_1, \ldots, u_k$
5. *normalize* $U$ so that rows have norm 1
6. consider the i-th row of $U$ as point $y_i \in \mathtt{R}^k$, $i = 1, \ldots, n$
7. cluster (kmeans) the points $\{y_i\}_{i=1,\ldots,n}$ into clusters $C_1, \ldots, C_k$

**Output:** clusters $A_1, \ldots A_k$
This algorithm is similar to Algorithm 1. The difference is in
- step 2 we have *normalized laplacian* instead of the *unnormalized laplacian*

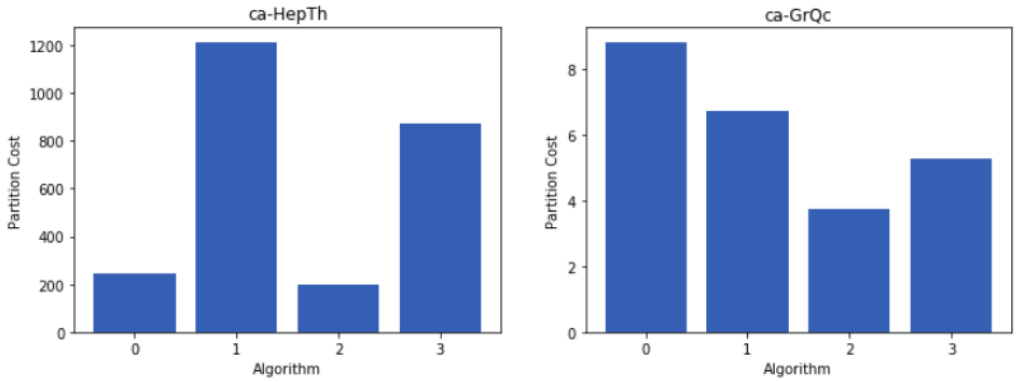
$$L' := I - D^{-1/2}AD^{-1/2}$$

- step 5 we *normalize* $U$

# 3 Performance

We will use the following graphs to access the performance of the algorithms.

`ca-HepTh ca-GrQc`




| Algo | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| `ca-HepTh` | 248.3 | 1214.0 | 198.8 | 871.9 |
| `ca-GrQc` | 8.8 | 6.7 | 3.7 | 5.26 |

For both graphs, algorithm 3 has the lowest partition cost hence has the most balanced parition. We will use Algorithm 3 : normalized spectral clustering (generalized eigenproblem, normalized U) to cluster our graph and produce the results.