APPENDICES

Contents

A Full code listing

Basic aim:

A program for modularization of a 100-node random planar network.

Challenge aim:

Regarding r = 2, the program of transforming the random planar network of different nodes into an empire network with a fixed number of interval vertices and modularizing it.

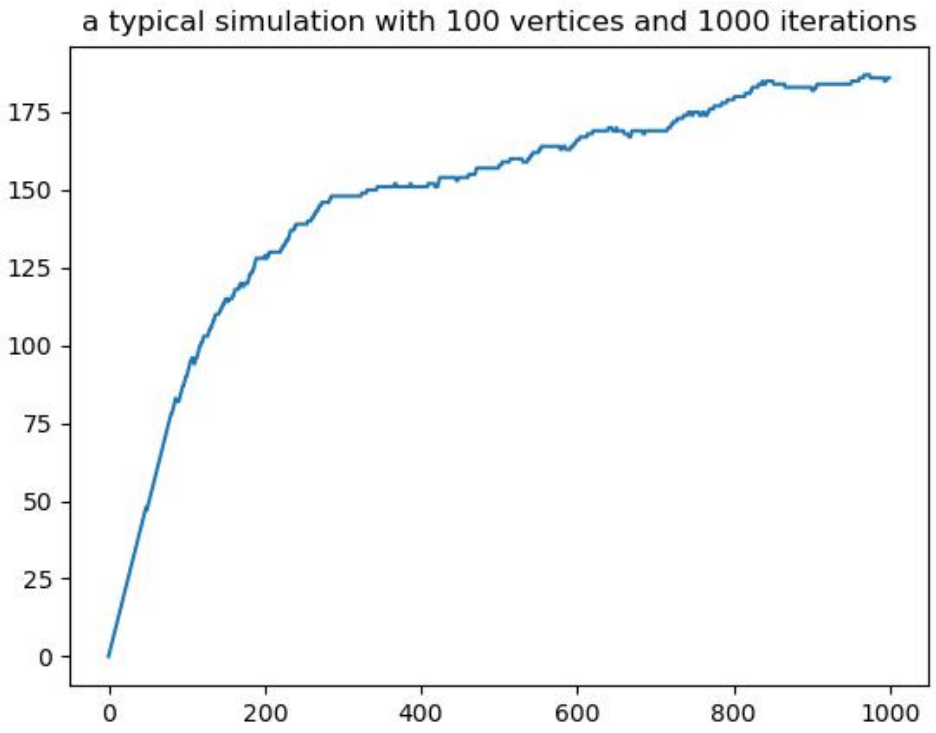
Regarding r = 2, the program of transforming the random planar network of different nodes into an empire network with a fixed number of interval vertices and modularizing it.

Regarding r = 2, the program of transforming the random planar network of different nodes into an empire network with a fixed number of interval vertices and modularizing it.

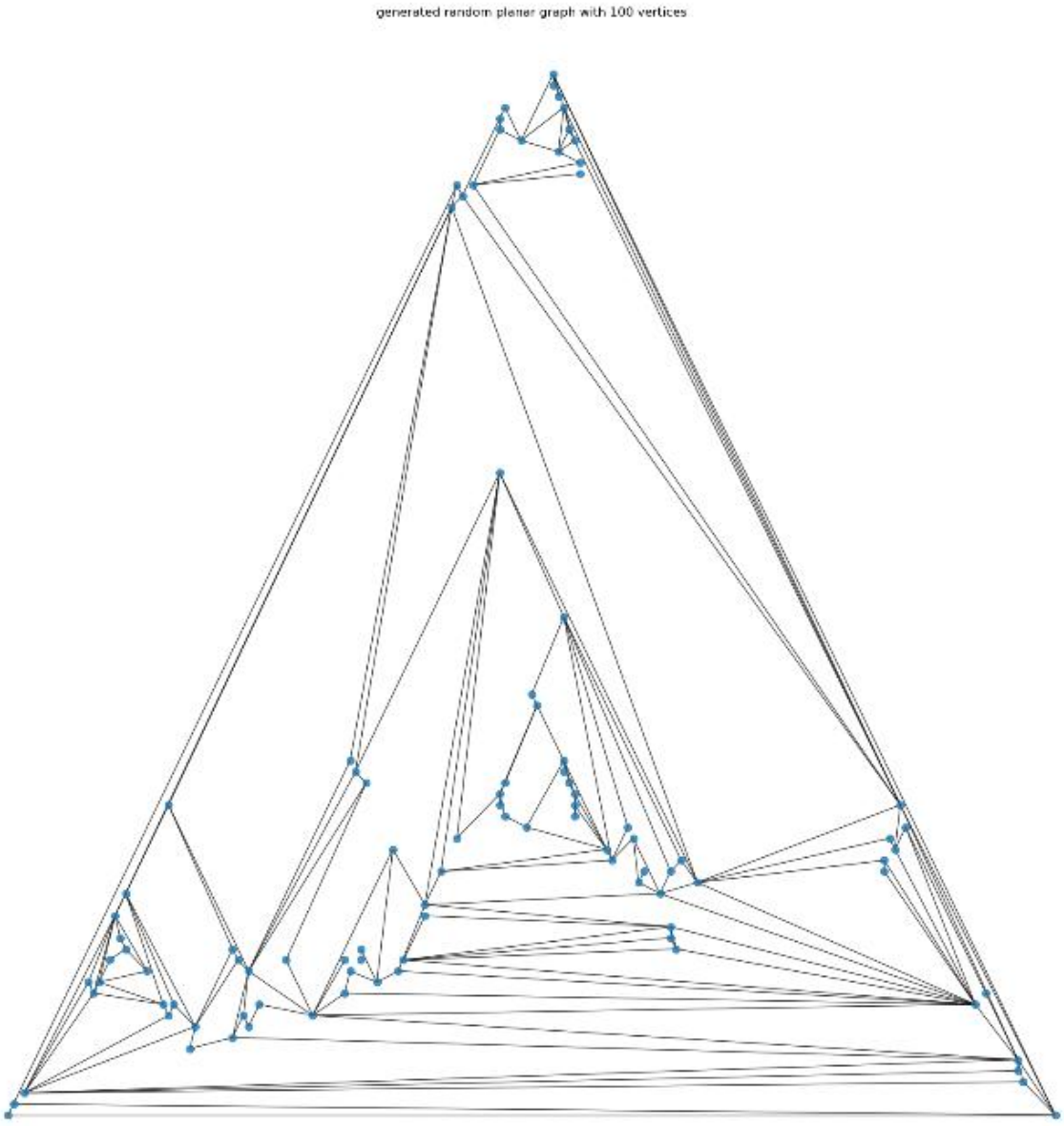
B Screen shots of sample runs

Basic aim:

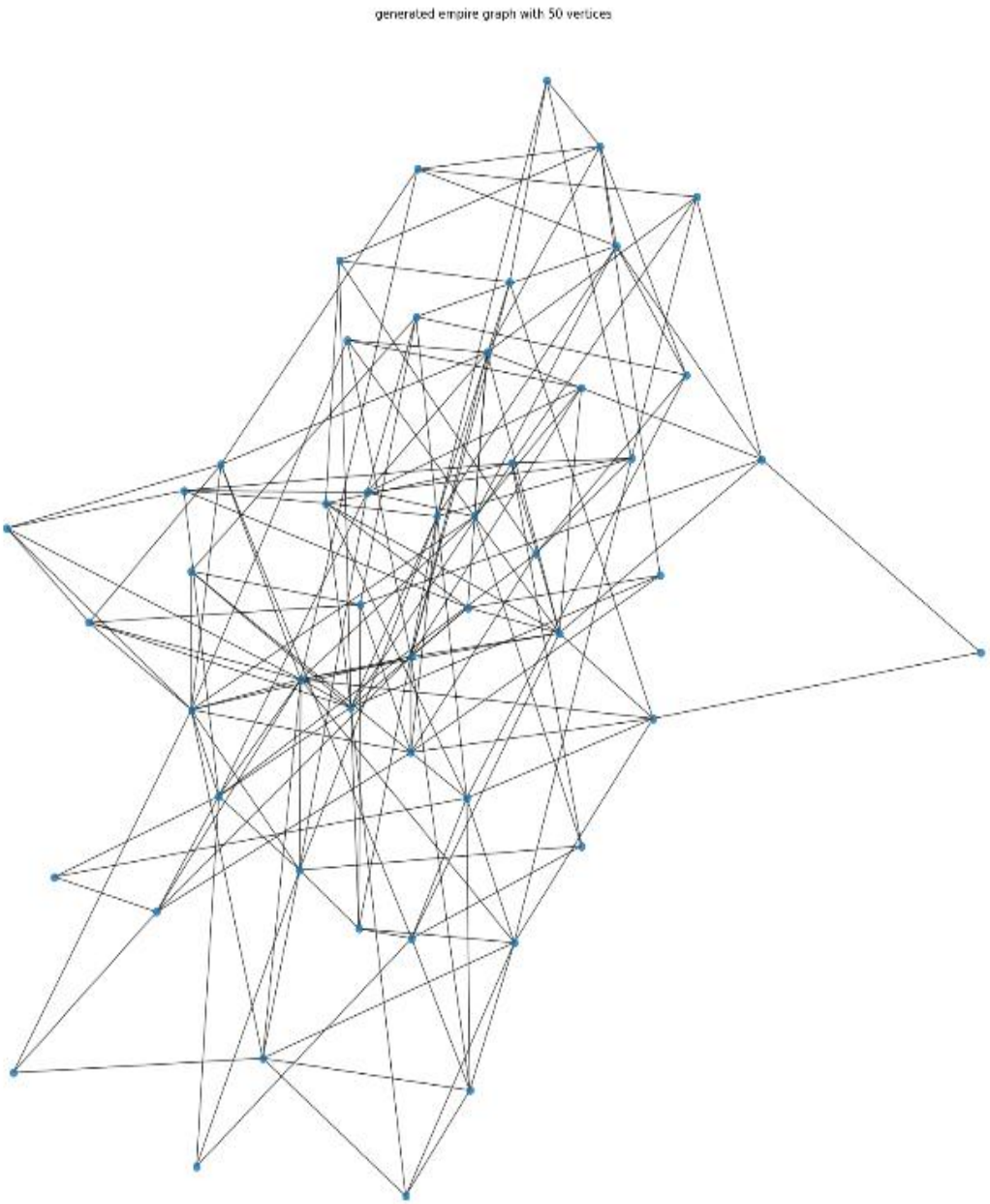
Then the simulation of the number of nodes and the number of iterations in the random planar network.



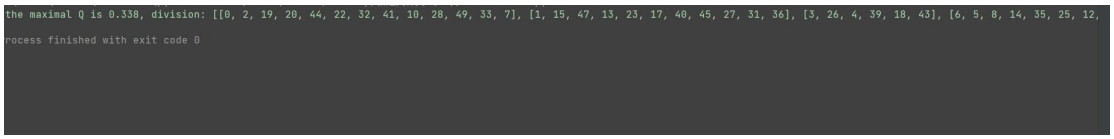
Generate the random planar network of 100 nodes.



In the case of r=2, a 50-node empire network is generated.



The 50-node empire network is divided into the community structure and the maximum Q value is the value of modularity.



Challenge aim:

r = 2: the average Q is 0.449, nodes = 200, vetices = 100.
r = 2: the average Q is 0.546, nodes = 400, vetices = 200.
r = 2: the average Q is 0.617, nodes = 600, vetices = 300.
r = 2: the average Q is 0.635, nodes = 800, vetices = 400.
r = 2: the average Q is 0.679, nodes = 1000, vetices = 500.
r = 2: the average Q is 0.700, nodes = 1200, vetices = 600.
r = 2: the average Q is 0.710, nodes = 1400, vetices = 700.
r = 2: the average Q is 0.717, nodes = 1600, vetices = 800.
r = 2: the average Q is 0.744, nodes = 1800, vetices = 900.
r = 2: the average Q is 0.751, nodes = 2000, vetices = 1000.

r = 3: the average Q is 0.339, nodes = 300, vetices = 100.
r = 3: the average Q is 0.426, nodes = 600, vetices = 200.
r = 3: the average Q is 0.455, nodes = 900, vetices = 300.
r = 3: the average Q is 0.491, nodes = 1200, vetices = 400.
r = 3: the average Q is 0.508, nodes = 1500, vetices = 500.
r = 3: the average Q is 0.512, nodes = 1800, vetices = 600.
r = 3: the average Q is 0.528, nodes = 2100, vetices = 700.
r = 3: the average Q is 0.551, nodes = 2400, vetices = 800.
r = 3: the average Q is 0.559, nodes = 2700, vetices = 900.
r = 3: the average Q is 0.565, nodes = 3000, vetices = 1000.

r = 5: the average Q is 0.237, nodes = 500, vetices = 100.
r = 5: the average Q is 0.295, nodes = 1000, vetices = 200.
r = 5: the average Q is 0.346, nodes = 1500, vetices = 300.
r = 5: the average Q is 0.334, nodes = 2000, vetices = 400.
r = 5: the average Q is 0.349, nodes = 2500, vetices = 500.
r = 5: the average Q is 0.367, nodes = 3000, vetices = 600.
r = 5: the average Q is 0.374, nodes = 3500, vetices = 700.
r = 5: the average Q is 0.369, nodes = 4000, vetices = 800.
r = 5: the average Q is 0.385, nodes = 4500, vetices = 900.
r = 5: the average Q is 0.402, nodes = 5000, vetices = 1000.

C Details of test data

We generate a large amount of experimental data and test data, including a large amount of iterative data and simulation data corresponding to the number of random planar network nodes, and data on the number of vertices of the empire network merged according to the nodes of the random planar network. And the data of the community division and the maximum modularity value after modularization, and the data of the random network average Q value of a large number of experiments with different numbers of nodes.

E A user guide to installation

I used Python 3.8 (base) version and brought in networkx, numpy, copy, sys, matplotlib.pyplot, os modules.

F usage of the software

It is recommended to use modern programming software such as Pycharm that I use, and not to use Python IDE software, because the amount of experiments in this project is huge, and the software may not fully meet the requirements of this project. I personally use the windows 10 operating system, and all items may have slight differences on different versions and different operating systems.

G full design diagrams

G.1 Project plan of date:

Week	Activity
First week	Background reading literature review.
Second week	Background reading literature review.
Third week	Background reading literature review.
Fourth week	Project specification and initial design.
Fifth week	Project specification and initial design.
Sixth week	Project specification and initial design.
Seven week	Software implementation and testing.
Eighth week	Software implementation and testing.
Ninth week	Software implementation and testing.
Tenth week	Software implementation and testing.
Eleventh week	Software experimentation and analysis of results.
Twelfth week	Software experimentation and analysis of results.
Thirteenth week	Software experimentation and analysis of results.
Fourteenth week	Software experimentation and analysis of results.
Fifteenth week	Write up of dissertation.

Sixteenth week	Write up of dissertation.
Seventeen week	Write up of dissertation.

G.2 Detailed introduction of the design time plan

In the early stages of project realization, we learned about the main materials of the project and expanded our knowledge about the project domain, combined with the existing materials to critically think about the methods in the project and added individual independent thinking, so that we have a general idea of the project direction, idea.

In the first week, we compare the algorithm of the theme material with other algorithms to determine the feasibility and reliability of the theme material, and conduct a preliminary design attempt.

In the second week, we began to make specific preliminary plans for the random plane network design, and at the same time transform the relevant knowledge of the Empire network to understand how to fit the theme materials, integrate your own design ideas into it, and complete the random plane network matching the Empire network. Conversion design.

In the third week, I learned a lot of modular network knowledge, starting from the most basic learning, such as what is a modular network, what is the role of modularity, why the network should be

modularized, and how to transform a random plane network to modularize the imperial network , Thinking about the role of the Empire Network.

In the fourth week, a large number of modular algorithms were compared, including Louvain algorithm, Girvan-Newman algorithm , fast Newman algorithm , fast Newman algorithm (update), etc. Finally, after extensive comparison, we chose the main material algorithm fast Newman and carried out the original design.

After completing the original design of the project, we carried out a series of programming, organized the structure of each network to be processed, and then after repeated comparison and research, we proposed a preliminary program design and improved the deficiencies of the original design. And expanded the modular publicity content, and then after repeated discussions with the supervisor, it was determined that in the experiment, the interval of the number of vertices of the empire network was kept unchanged, and then the fixed-point interval of the plane network was changed according to the value of r, and a Relatively complete project design version.

In the seventh and eighth weeks, complete the correction of the original design of the project, write the algorithm and pseudo-code, and conduct the preliminary structure design of the program.

In the ninth week, the program design plan is initially completed and tested, and then the code is run to check whether the three algorithms of the basic program framework are executed coherently, whether they can run normally, whether the deviation of the experimental results is within the acceptable range, that is, whether the Q value is (- 0.5, 1).

In the tenth week, discuss the shortcomings of the original plan and the areas that need to be modified with the supervisor, repeatedly debug the program errors, make the program run normally, and ensure the correctness of the experiment.

Finally, in the final stage of the project, we ran a large number of experiments and obtained relatively reliable experimental results. We analyzed the modularity distribution, and carried out statistical research on the distribution of modularity values. After completing the challenge-level tasks, we obtained relatively reliable experimental results. A comprehensive summary of the discovered phenomena and laws was carried out, and all the steps including the design, operation, and analysis of experimental results of the program and project were completed, all tasks were perfectly realized, the existing and known algorithms were studied in practice, and the thesis was completed.

In the eleventh week, complete the programming and operation of the basic tasks, and try to run challenging experiments, conduct a

large number of experimental studies, fully verify the correctness and operability of the program after completing the basic tasks, and complete the modularity value Calculate and complete the community division (under the maximum Q value).

A large number of experiments were completed in the twelfth and thirteenth weeks, and the average modulus distribution results of the large random planar network were obtained. The experimental results are analyzed and compared, and the distribution law of modularity of random planar network under different r values is obtained, and its characteristics are counted and studied.

From the fourteenth week to the end of the project, complete the thesis and procedures, meet the project requirements, and complete the basic tasks and challenges.

G.3 Pseudo-code:

Pseudo-code of generate the random planar graph:

Random planar graph: initially G is the empty graph on vertices_num vertices, repeat iter_num times: choose position (i,j), if G contains {i,j} remove it, else add {i,j}, if it keeps the graph planar, and got a planar graph (vertices_nums).

The algorithm of generate the empire graph:

Let B = (Bi)i=1,...,n/r be the partition of a set of labelled vertices into blocks B_1, B_2,..., such that B_k contains vertices $(k-1)_r + 1$,$(k-1)_r + 2$,..., k_r. For each i 2 {1,...,r}, we will denote by k_i the vertex labelled $(k-1)_r + i$ in B_k. Note that, for r = 1, the blocks contain just a single vertex, and, for r > 1 we always assume (even when we don't state it explicitly) that n/r is an integer. From now on an r-empire graph will be a pair (G, r) where G is a planar graph generated at random on the given set of vertices, and r is a positive integer, with r <= n and such that n/r is a positive integer giving the size of the blocks in B, as described above. In fact, we will work with another graph S = S(G), the skeleton of (G,

r). The vertex set of S has n/r vertices, one for each block B_i, and there is an edge joining vertex i to j (for i = j) if and only if there is at least one edge in G between a vertex in B_i and a vertex in B_j (we ignore loops, i.e. edges connecting two vertices in the same block).

Pseudo-code of generate the Empire graph:

Create n/r vertices of the empire graph,for each vertex v in planar graph:list all connections {v,u}add edge to empire graph between the empire of v and that of u.

Pseudo-code of fast algorithm for detecting community structure in networks.

1. Regard each vertex in the list of the generated empire graph as a community, and form an adjacency matrix E = (eij) through each node.

2. The modularity of a partition of the vertex set is $Q = \Sigma_i (e_{ii}-a_i^2)$

(1) e_{ii} represents the ratio of the number of edges within community i to the number of edges in the entire network,

Also e_{ij} is the fraction of edges that connect nodes in group i to nodes in group j.

(2) ai represents the proportion of edges connected to nodes in community i, $a_i = \Sigma_j e_{ij.}$

In fact, comparing these definitions with the treatment of the same algorithm in Newman's Networks book it turns out that it is better to define a_i as. $\frac{e_{ii}}{m} + \sum_{v \in Ci} \frac{deg\ v}{2m} = e_{ii} + \sum_j e_{ij}/2$

(3) Calculate the value of Q for the combination of the two communities

3. Find the maximum increase and minimum decrease of Q to merge

(1) compute ΔQ = 2(eij − aiaj), in time O(m) (in fact one should write ΔQij as there is one

value for each pair of communities).

(2) Update eij for the corresponding elements and sum the rows and columns associated with

the community, O(n)

(3) The total complexity is O(m + n)

4. Repeat the second step to merge the entire network into a community to get the largest Q value. The maximum number of merging is n − 1, O((m + n)n), O(n2).

G.4 Design drawings and other design documents:

2 A Markov chain algorithm

Let $G = (V, E)$ be any simple graph. We define a Markov chain $M(G)$ with state space all planar subgraphs of G and with transitions defined as follows.

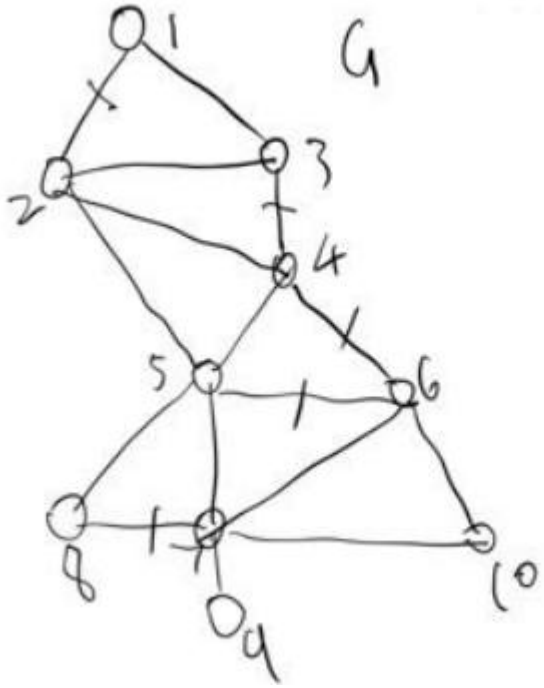
A *position* of G consists of an unordered pair of distinct vertices of G. If X_t denotes the state $M(G)$, at time t, then X_{t+1} is chosen as follows. A position f of G is chosen uniformly at random.

(a) If the position f contains an edge e of X_t then $X_{t+1} = X_t \backslash e$.

(b) If the position $f = (i, j)$ does not contain an edge in X_t then X_{t+1} is formed from X_t by adding an edge (i, j) provided this addition preserves planarity,

(c) otherwise $X_{t+1} = X_t$.

It is easy to verify that (X_t) is an irreducible aperiodic Markov chain whose transition matrix is symmetric. Thus, X_t has a limiting stationary distribution which is uniform over the set of planar subgraphs of G. In principle therefore it gives an easily implemented algorithm for generating a planar subgraph of G which will be approximately uniformly at random. The closeness of the approximation will be governed by the mixing rate of the chain, and this will depend on the graph G. In particular, when $G = K_n$ it gives what appears to be a fairly effective way of generating a random planar graph.


1
G
2
3
4
5
6
8
9
10

Let $B = (B_i)_{i=1,\ldots,n/r}$ be the partition of a set of labelled vertices into blocks $B_1, B_2, \ldots$, such that B_k contains vertices $(k-1)r+1, (k-1)r+2, \ldots, kr$. For each $i \in \{1, \ldots, r\}$, we will denote by k_i the vertex labelled $(k-1)r+i$ in B_k. Note that, for $r = 1$, the blocks contain just a single vertex, and, for $r > 1$ we always assume (even when we don't state it explicitly) that n/r is an integer. From now on an r-empire graph will be a pair (G, r) where G is a planar graph generated at random on the given set of vertices, and r is a positive integer, with $r \leq n$ and such that n/r is a positive integer giving the size of the blocks in B, as described above. In fact, we will work with another graph $S = S(G)$, the skeleton of (G, r). The vertex set of S has n/r vertices, one for each block B_i, and there is an edge joining vertex i to j (for $i \neq j$) if and only if there is at least one edge in G between a vertex in B_i and a vertex in B_j (we ignore loops, i.e. edges connecting two vertices in the same block).

Assortative mixing as defined in [New03] is "the tendency for vertices in networks to be connected to other vertices that are like or unlike them in some way." Here's a parameter that provides a measure for assortativity. The modularity of a graph G w.r.t. a vertex partition $\mathcal{C}$ of V is defined as (this is taken, for instance, from my COMP324 slides, $e(c)$ denotes the number of edges in $G[c]$.):

$$Q_{\mathcal{C}}(G) = \frac{1}{m} \sum_{c \in \mathcal{C}} e(c) - \frac{1}{4m^2} \sum_{c \in \mathcal{C}} \left(\sum_{i \in c} \deg_G i \right)^2 .$$

Remarks.

1. The definition only makes sense if $m > 0$.
2. $Q_{\mathcal{C}}(G) \in [-1, 1]$.

   The term $m^{-1} \sum_{c \in \mathcal{C}} e(c) \in [0, 1]$. The term $(2m)^{-2} \sum_{c \in \mathcal{C}} \left(\sum_{i \in c} \deg_G i \right)^2$ is also in $[0, 1]$ as $4m^2$ is the square of the sum of all degrees in G.
3. The modularity depends on TWO components. It is higher when many of the edges of G fall within classes (because of the first term in it), but it is reduced by a quantity that depends on the degrees within clusters (see Figure 3 for an example.) Interestingly this term is NOT simply measuring how much of the degree of any v is "used" within the same cluster.

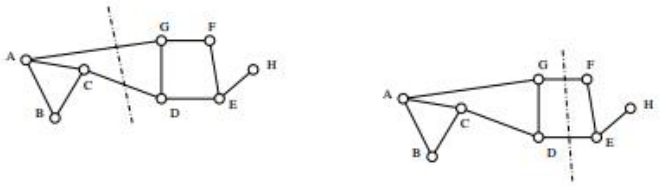


Figure 3: Two partitions in a small graph. The graph has $m = 10$ edges, and in both cases $\sum_{c \in \mathcal{C}} e(c) = 8$. However the sum of the squares of the degree sums in the classes are $(2+3+3)^2 + (1+2+3\times3)^2 = 208$ for the graph on the left-hand side, and $(2+4\times3)^2 + (1+2+3)^2 = 232$ for the other one. Hence the partition on the left-hand side has slightly larger modularity.

In this project we are interested in computing

$$Q(S) = \max_{\mathcal{C}} Q_{\mathcal{C}}(S)$$

where S is the skeleton of a given r-empire graph.

Our algorithm is based on the idea of modularity. Given any network, the GN community structure algorithm always produces *some* division of the vertices into communities, regardless of whether the network has any natural such division. To test whether a particular division is meaningful we define a quality function or "modularity" Q as follows [6]. Let e_{ij} be the fraction of edges in the network that connect vertices in group i to those in group j, and let $a_i = \sum_j e_{ij}$ [19]. Then

$$Q = \sum_i (e_{ii} - a_i^2) \tag{1}$$

Since the joining of a pair of communities between which there are no edges at all can never result in an increase in Q, we need only consider those pairs between which there are edges, of which there will at any time be at most m, where m is again the number of edges in the graph. The change in Q upon joining two communities is given by $\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$, which can clearly be calculated in constant time. Following a join, some of the matrix elements e_{ij} must be updated by adding together the rows and columns corresponding to the joined communities, which takes worst-case time $\mathrm{O}(n)$. Thus each step of the algorithm takes worst-case time $\mathrm{O}(m + n)$. There are a maximum of $n - 1$ join operations necessary to construct the complete dendrogram and hence the entire algorithm runs in time $\mathrm{O}((m+n)n)$, or $\mathrm{O}(n^2)$ on a sparse graph. The algorithm has the added advantage of calculating the value of Q as it goes along, making it especially simple to find the optimal community structure.

1. Regard each vertex in the list of the generated empire graph as a community, and form an adjacency matrix $E = (e_{ij})$ through each node.
2. The modularity of a partition of the vertex set is

$$Q = \sum_i (e_{ii} - a_i^2)$$

(1) e_{ii} represents the ratio of the number of edges within community i to the number of edges in the entire network,

Also e_{ij} is the fraction of edges that connect nodes in group i to nodes in group j.

(2) a_i represents the proportion of edges connected to nodes in community i, $a_i = \sum_j e_{ij}$,

In fact, comparing these definitions with the treatment of the same algorithm in Newman's Networks book it turns out that it is better to define a_i as

$$\frac{e_{ii}}{m} + \sum_{v \in C_i} \frac{\deg v}{2m} = e_{ii} + \sum_j e_{ij}/2$$

(3) Calculate the value of Q for the combination of the two communities

3. Find the maximum increase and minimum decrease of Q to merge

(1) compute $\Delta Q = 2(e_{ij} - a_i a_j)$, in time $O(m)$ (in fact one should write ΔQ_{ij} as there is one value for each pair of communities).

(2) Update e_{ij} for the corresponding elements and sum the rows and columns associated with the community, $O(n)$

(3) The total complexity is $O(m + n)$

4. Repeat the second step to merge the entire network into a community to get the largest Q value. The maximum number of merging is $n - 1$, $O((m + n)n)$, $O(n^2)$.

Figure 1.4: Example graph to compute the modularity "paper and pencil".

EXAMPLE.

The first time there are $n = 5$ communities, the n vertices.
Here are the values of a_i and e_{ij}

	1	2	3	4	5	a_i
1	0	1/5	0	0	1/5	2/10
2	1/5	0	1/5	1/5	1/5	4/10
3	0	1/5	0	0	0	1/10
4	0	1/5	0	0	0	1/10
5	1/5	1/5	0	0	0	2/10

The changes ΔQ_{ij} now (as in 3.(1) above) and in thick black the maximum values:

		2	3	4	5
1	2(1/5-8/100)	2(0-2/100)	2(0-2/100)	**2(1/5-4/100)**	
2		**2(1/5-4/100)**	**2(1/5-4/100)**	2(1/5-8/100)	
3			2(0-1/100)	2(0-2/100)	
4				2(0-2/100)	

So we have three equal options: [[1,5][2][3][4]], or [[1][2,3][4][5]], or even [[1][2,4][3][5]]. Note that the last two are isomorphic, they differ only in the labelling of the community elements. So in what follows we show two options (note that the actual program will use a rule defined by Daozheng to select only one option).

OPTION 1: [[1,5][2][3][4]]

Here are the values of a_i and e_{ij}

	[1,5]	2	3	4	a_i
[1, 5]	1/5	2/5	0	0	4/10
2	2/5	0	1/5	1/5	4/10
3	0	1/5	0	0	1/10
4	0	1/5	0	0	1/10

The changes ΔQ_{ij} now:

	2	3	4
[1,5]	**2(2/5-16/100) = 0.24**	2(0-4/100)	2(0-4/100)
2		2(1/5-4/100)	2(1/5-4/100)
3			2(0-1/100)

So the new communities are [[1,2,5],[3],[4]].

One more time. The a_i and e_{ij}:

	[1,2,5]	3	4	a_i
[1, 2, 5]	3/5	1/5	1/5	8/10
3	1/5	0	0	1/10
4	1/5	0	0	1/10

and the ΔQ_{ij}

	3	4
[1,2,5]	**2(1/5-8/100)**	**2(1/5-8/100)**
3		2(0-1/100)

There are still positive changes in modularity, therefore in this example Newman's algorithm separates one of the two vertices of minimum degree from the rest.

OPTION 2: [[1][2,3][4][5]]

In this case the values of a_i and e_{ij} are

	1	[2,3]	4	5	a_i
1	0	1/5	0	1/5	2/10
[2, 3]	1/5	1/5	1/5	1/5	5/10
4	0	1/5	0	0	1/10
5	1/5	1/5	0	0	2/10

and the changes ΔQ_{ij}

	[2,3]	4	5
1	2(1/5-10/100)	2(0-2/100)	**2(1/5-4/100)=0.16**
[2, 3]		2(1/5-5/100)	2(1/5-10/100)
4			2(0-2/100)

So the new communities are [[1,5],[2,3],[4]].

I stop here with OPTION 2 ... but it is clear that, in this example, this will lead to the same solution as OPTION 1 (in one more step we would choose to aggregate [1,5] and [2,3] ...). Of course this convergence of the different options is not general: in other examples it may not happen.