# Contents

# 1 Appendix

## 1.1 Network implementation

In order to efficiently implement networks and their analysis on a computer, it is necessary to use data structures adjusted to these problems. A short and transparent introduction to data structures and algorithms is in the book of Skiena (Skiena, 2008). In this section, we discuss some essential data structures appropriate for network analysis and give a brief description of fundamental algorithms. The purpose of this section is not to list different algorithms and source code, but rather to sketch the basic ideas behind the data structures and algorithms. For source code of data structures and algorithms, the reader is encouraged to the lecture of (Skiena, 2008) and (Merali, 2010).

**Matrix implementation.** To begin with, we consider the implementation of adjacency matrices as introduced in section **??**. Matrix implementations work also for the other matrices introduced in section **??**. All adjacency matrices are by definition square matrices. Their entries are either 0 or 1, and can take any floating number value for weighted networks. In this work, we neglect negative edge weights. The number of nodes in most complex network datasets is relatively large. Starting with small networks (100 nodes, conference contacts (Isella et al., 2011)), complex networks can be gigantic (0.5 billion nodes, twitter tweeds (Yang and Leskovec, 2011)). Note that the size of the adjacency matrices scales with the square of the networks size, hence large networks intractable for straightforward computer-based matrix analyses.

Nevertheless, there is one feature, that most adjacency matrices of real-world networks have in common: they are *sparse*, i.e. the vast majority of their entries are zeros[1]. Since zeros do not contribute to matrix operations as products or additions, it is reasonable to use data structures ignoring zeros. These data structures are called `sparse matrices`. Their advantages is (1) they save much memory and (2) computations are faster, because operations with zeros involved are not executed. Sparse matrix data structures are available in most modern computer languages (e.g. Matlab, Python: `scipy` library, C/C++: `boost` library). They perform well for all problems based on adjacency matrices, e.g. degree or eigenvector centrality. However, matrix methods are not suitable for the computation of many other network measures, such as betweenness, closeness or network navigation.

[1]Typically, the number of edges in the network is of the same order as the number of nodes.

**Graph implementation.** The weakness of matrix representations of networks is that it is rather complicated to *traverse* a network using matrices. A traversal is a procedure like: start at a node, visit all of its neighbors, from each neighbor visit its neighbor and so forth, until there are no more new nodes to traverse. This is a searching process. These processes are used in many implementations of graph theoretic methods.

An alternative implementation to the adjacency matrix is an *adjacency list*. It stores the neighbors of every node and is implemented in terms of a linked list. Using the example network of **??**, we get the following adjacency list:

$$\begin{aligned}1 &\to 2, 3\\ 2 &\to 4\\ 3 &\to 2\\ 4 &\to 2, 3.\end{aligned}$$

In order to traverse the graph starting at node 1, we can choose any of the neighbors of 1 and repeat the process until we have traversed all nodes. One possible traversal starting at 1 would be $1 \to 3 \to 2 \to 4$.

During a traversal process, one can decide to either exploit the whole neighborhood of a node first and then traverse the next generation or choose a neighbor of every traversed node at every step. These two essential searching processes are called breadth-first-search (BFS) and depth-first-search (DFS), respectively. The difference between the two lies in the order of traversed nodes. Figure 1.1 shows resulting search trees of the two methods. Starting at node 1, the traversal $1 \to 3 \to 2 \to 4$ would be found using a DFS-search, while a BFS-search would yield $1 \to 2 \to 3 \to 4$. It should be noted that in general there exist multiple BFS and DFS trees for each starting node.

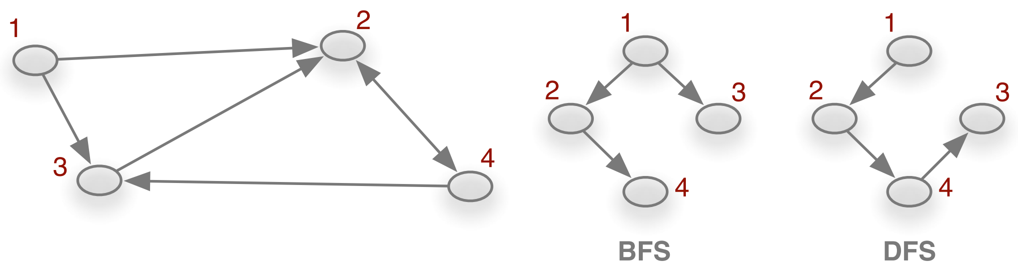

**Figure 1.1.** Breadth-first-search and depth-first-search trees in the directed network of fig. **??**. Searches are started at node 1.

Both search algorithms are used in many applications. BFS is efficient to compute shortest paths in unweighted networks. With every generation in a BFS tree, the distance from the starting node is incremented by 1, and thus the set of nodes with a certain distance from the starting node can be directly read from the BFS tree (see figure 1.1). Shortest paths in weighted networks can be identified using a the algorithm of Dijkstra (Dijkstra, 1959). DFS can be used to identify connected components in directed graphs (see next section).

Implementations of graph structures as discussed above are for example available in the libraries `networkx` (Python), `igraph` (C, Python, R), `Lemon` and `Boost` (C++).

**Hard problems.** Although the libraries introduced above provide a huge and efficient toolbox for network analysis, there are still network problems, where no efficient algorithm is known for their exact solution. In the language of complexity theory, the time to solve these problems scales with the problem size in non-polynomial time. Problems of this kind can typically be solved exactly only for small system sizes.

Probably the most popular example is the *traveling salesman problem*: a salesman has to traverse a set of cities and thereby choose the order of those cities that minimizes the total distance. For small problem sizes, it is possible just to try out all possible combinations and find the minimal total distance. The number of possible combinations, however, grows factorial with the system size, i.e. finding a solution takes $t \propto n!$ for $n$ cities. In other words, if the problem could be solved for 20 cities in 1 second, it would take 21 seconds to solve it for 21 cities, 7 minutes for 22 cities and 3 million years for 30 cities!

A more exhaustive overview about hard problems is in (Skiena, 2008) and the references therein. Generally, heuristic methods have to be used in order to get an approximate solution. It should be noted that the *maximum clique* problem (section xx) and *graph partitioning* (see section xx, (Brandes et al., 2007)) belong to the class of hard problems.

## 1.2 Subgraphs and maximum modularity

We derive an estimation of the maximum modularity value depending on the number of modules in the network. The results are derived for a clique of modules, but remain the same for a ring of modules as it is reasonable in finite systems (see below). In addition, the estimation is also valid for directed networks (see below).

The modularity of a network can be computed using the equation

$$Q = \sum_i \left( e_{ii} - a_i^2 \right) , \tag{1.1}$$

where $e_{ij}$ is the fraction of edges pointing from community $i$ to community $j$. The last term corresponds to the fraction of all edges that are connected to community $i$, i.e.

$$a_i = \sum_j e_{ij} .$$

Since the sum over all edge fraction has to be 1, it is $\sum_{ij} e_{ij} = 1$. If a network consists of two modules $x$ and $y$, the fraction of edges in $y$ is

$$y = c - x, \tag{1.2}$$

where the constant $c < 1$ is the fraction of all inner module edges. In general, it is $c = \mathrm{Tr}(e)$.

### 1.2.1 Two modules

In the case of two communities, the fraction of inter-module-edges is uniquely determined by the fraction of inner-module-edges.

The matrix $e_{ij}$ takes the form

$$e_{ij} = \begin{pmatrix} x & \frac{1}{2}(1-x-y) \\ \frac{1}{2}(1-x-y) & y \end{pmatrix},$$

where $x$, $y$ are the edge fractions *in* communities 1 and 2 and $\frac{1}{2}(1-x-y)$ is the fraction of edges *between* communities 1 and 2. The corresponding expression for $Q$ is.

$$Q = x - \left(x + \frac{1-x-y}{2}\right)^2 + y - \left(y + \frac{1-x-y}{2}\right)^2.$$

This function does not possess a maximum over the total domain, but there is a maximum in the subdomain $0 < x < 1$, $0 < y < 1$. Condition (1.2) yields

$$Q = \frac{1}{2} + 2\,cx - 2\,x^2 - \frac{1}{2}\,c^2 + c.$$

Using condition (1.2) restricts the function to tuples $(x, y)$, where $x + y = c$, which corresponds to a line $y = c - x$. Thus, we are looking for the maximum along this line using the condition

$$\frac{\partial Q}{\partial x} = 2c - 4x = 0.$$

It follows $x = c/2$ and the maximum condition $\partial^2 Q/\partial x^2 = -4 < 0$ is satisfied. Using (1.2) gives the solution

$$x = \frac{c}{2}, \quad y = \frac{c}{2}. \tag{1.3}$$

The corresponding modularity is

$$\begin{aligned} Q &= \frac{c}{2} - \frac{1}{4}\left(1 + \frac{c}{2} - \frac{c}{2}\right)^2 + \frac{c}{2} - \frac{1}{4}\left(1 + \frac{c}{2} - \frac{c}{2}\right)^2 \\ &= c - \frac{1}{2}. \end{aligned}$$

The case where a maximum fraction of edges is in the modules and a minimum fraction is between modules is met, if $c \to 1$. In this case, the modularity takes its maximum value. The limit is

$$\lim_{c\to 1} x = 1/2, \quad \lim_{c\to 1} y = 1/2, \quad \lim_{c\to 1} Q = 1/2. \tag{1.4}$$

For the case of two modules, the maximum modularity is found for two equally sized modules of approximate size 1/2. The maximum modularity is then $Q = 0.5$. We consider the case of more modules below.

### 1.2.2 Arbitrary number of modules

In the case of more than two modules, all modules can have different sizes in the first place and can be connected among themselves arbitrarily. The general module-matrix takes the form

$$e_{ij} = \begin{pmatrix} x_1 & & \dots & d \\ & x_2 & & \\ \vdots & & \ddots & \vdots \\ & & & \\ d & & \dots & x_n \end{pmatrix}. \tag{1.5}$$

All non-diagonal elements are

$$d = \frac{1 - \mathrm{Tr}\ (e)}{n(n-1)} = \frac{1-c}{n(n-1)}$$

with $c \equiv \mathrm{Tr}\ (e) = \text{const.} < 1$. Thus, the general expression for modularity is

$$Q = c - \sum_i \left( \sum_j e_{ij} \right)^2 . \tag{1.6}$$

We use the above expression for the non-diagonal elements $d$ and compute the expression $\sum_j e_{ij}$ in equation (1.6).

$$\sum_j e_{ij} = e_{ii} + \sum_{j \neq i} e_{ij} = x_i + (n-1)\frac{1-c}{n(n-1)} = x_i + \frac{1-c}{n}. \tag{1.7}$$

Now we insert $\sum_j e_{ij} = x_i + \frac{1-c}{n}$ in equation (1.6) and after some algebra we get the general expression for the modularity for networks of the form (1.5):

$$Q = c - \sum_i x_i^2 - \frac{1-c^2}{n} = \sum_i x_i - \sum_i x_i^2 - \frac{1 - (\sum_i x_i)^2}{n}. \tag{1.8}$$

In order to find the relevant maximum of (1.8), its slope has to vanish along a hyperplane defined by

$$\sum_i x_i = c = \text{const.} < 1. \tag{1.9}$$

Since $c$ is constant, the relevant part of (1.8) for the maximum is

$$\begin{aligned} Q_{\text{relevant}} \equiv Q_{\text{r}} = -\sum_{i=1}^{n} x_i^2 &= -\sum_{i=1}^{n-1} x_i^2 - \underbrace{\left(c - \sum_{i=1}^{n-1} x_i\right)^2}_{x_n^2} \\ &= -\sum_{i=1}^{n-1} x_i^2 - c^2 + 2c\sum_{i=1}^{n-1} x_i - \left(\sum_{i=1}^{n-1} x_i\right)^2 . \end{aligned}$$

Note that the sum on the r.h.s. runs to $n-1$. This effectively eliminates the last variable. The derivative of $Q$ is

$$\frac{\partial Q}{\partial x_i} = \frac{\partial Q_{\text{r}}}{\partial x_i} = -2\sum_{i=1}^{n-1} x_i + 2c(n-1) - 2(n-1)\sum_{i=1}^{n-1} x_i. \tag{1.10}$$

In order to find a maximum, the derivative has to vanish, i.e.

$$\begin{aligned} 0 &= -2\sum_{i=1}^{n-1} x_i + 2c(n-1) - 2(n-1)\sum_{i=1}^{n-1} x_i \\ &= -\sum_{i=1}^{n-1} x_i + c(n-1) - (n-1)\sum_{i=1}^{n-1} x_i \\ &= cn - c - n\sum_{i=1}^{n-1} x_i + \sum_{i=1}^{n-1} x_i - \sum_{i=1}^{n-1} x_i \\ &= cn - c - n\sum_{i=1}^{n-1} x_i . \end{aligned}$$

It follows

$$\underbrace{c - \sum_{i=1}^{n-1} x_i}_{x_n} = \frac{c}{n}.$$

Thus,

$$x_n = \frac{c}{n}. \tag{1.11}$$

Hence, the maximum of $Q$ is obtained, if all modules have the same size, i.e. $x_i = \frac{c}{n}\ \forall i$.

In order to find the maximum value of $Q$, we insert the module size $x_i = c/n$ into equation (1.8) and get

$$Q = c - \sum_{i=1}^{n} \left(\frac{c}{n}\right)^2 - \frac{1-c^2}{n} = c - \frac{c^2}{n} - \frac{1}{n} + \frac{c^2}{n}.$$

Thus, it follows for dense modules

$$\lim_{c\to 1} Q = 1 - \frac{1}{n}. \tag{1.12}$$

Consequently, the maximum value of $Q$ is determined by the number of modules. The same result was found using probabilistic arguments in (Good et al., 2010).

**Finite systems.** In finite systems, we get a minimum fraction of inter-module edges, when modules are connected to each other on a ring, each module having two nearest neighbors. In this case we set $e_{ij} = \frac{1}{n}(1-c)$ for $j = i+1$ and $j = i-1$ and all other elects are zero. This yields

$$e_{ij} = \begin{pmatrix} x_1 & \frac{1}{n}(1-c) & & \cdots & & 0 \\ \frac{1}{n}(1-c) & x_2 & \frac{1}{n}(1-c) & & & \\ & \frac{1}{n}(1-c) & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ & & & \ddots & x_{n-1} & \frac{1}{n}(1-c) \\ 0 & & & \cdots & \frac{1}{n}(1-c) & x_n \end{pmatrix}. \tag{1.13}$$

It follows immediately that $\sum_j e_{ij} = x_i + \frac{1-c}{n}$, which is exactly (1.7). Inserting this into equation (1.6) gives the same expression for modularity (1.8) as for the general case:

$$Q = c - \sum_i x_i^2 - \frac{1-c^2}{n} = \sum_i x_i - \sum_i x_i^2 - \frac{1-(\sum_i x_i)^2}{n}.$$

Thus, the results remain unchanged for modules along a chain.

**Directed networks.** In analog to equation (1.1) the modularity of directed networks can be written as (Kao et al., 2007)

$$Q = \sum_i e_{ii} - a_i^{\text{in}} a_i^{\text{out}}. \tag{1.14}$$

where

$$a_j^{\text{in}} = \sum_i e_{ij} \qquad \text{and} \qquad a_i^{\text{out}} = \sum_j e_{ij}.$$

The structure of the inter-module edges takes the form of the matrix (1.13) and thus results do not differ either for the directed case.

# Bibliography

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2007). On Finding Graph Clusterings with Maximum Modularity. In *Graph-Theoretic Concepts in Computer Science*, volume 4769, pages 121–132. Springer Berlin Heidelberg.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Good, B. H., de Montjoye, Y., and Clauset, A. (2010). The performance of modularity maximization in practical contexts. *Phys. Rev. E*, 81(4):046106.

Isella, L., Stehlé, J., Barrat, A., Cattuto, C., Pinton, J.-F., and Van den Broeck, W. (2011). What's in a crowd? Analysis of face-to-face behavioral networks. *J. Theor. Biol.*, 271(1):166–180.

Kao, R. R., Green, D. M., Johnson, J., and Kiss, I. Z. (2007). Disease dynamics over very different time-scales: foot-and-mouth disease and scrapie on the network of livestock movements in the UK. *Journal of the Royal Society Interface*, 4(16):907–916.

Merali, Z. (2010). Computational science: Error. Why scientific computing does not compute. *Nature*, 467:775–777.

Skiena, S. S. (2008). *The algorithm design manual.* Springer, London, 2nd edition.

Yang, J. and Leskovec, J. (2011). Patterns of temporal variation in online media. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 177–186, New York, NY, USA. ACM.