Community detection and visualization of networks with the map equation framework

Ludvig Bohlin, Daniel Edler, Andrea Lancichinetti, and Martin Rosvall[*]

Integrated Science Lab, Department of Physics, Umeå University, SE-901 87 Umeå, Sweden

Large networks contain plentiful information about the organization of a system. The challenge is to extract useful information buried in the structure of myriad nodes and links. Therefore, powerful tools for simplifying and highlighting important structures in networks are essential for comprehending their organization. Such tools are called community detection methods and they are designed to identify strongly intra-connected modules that often correspond to important functional units. Here we describe one such method, known as the map equation, and its accompanying algorithms for finding, evaluating, and visualizing the modular organization of networks. The map equation framework is very flexible and can with its search algorithm Infomap, for example, identify two-level, multi-level and overlapping organization in weighted, directed, and multiplex networks. Because the map equation framework operates on the flow induced by the links of a network, it naturally captures flow of ideas and citation flow, and is therefore well-suited for analysis of bibliometric networks.

This tutorial builds on a book chapter to be published in *Measuring scholarly impact: theory and practice* (Springer).

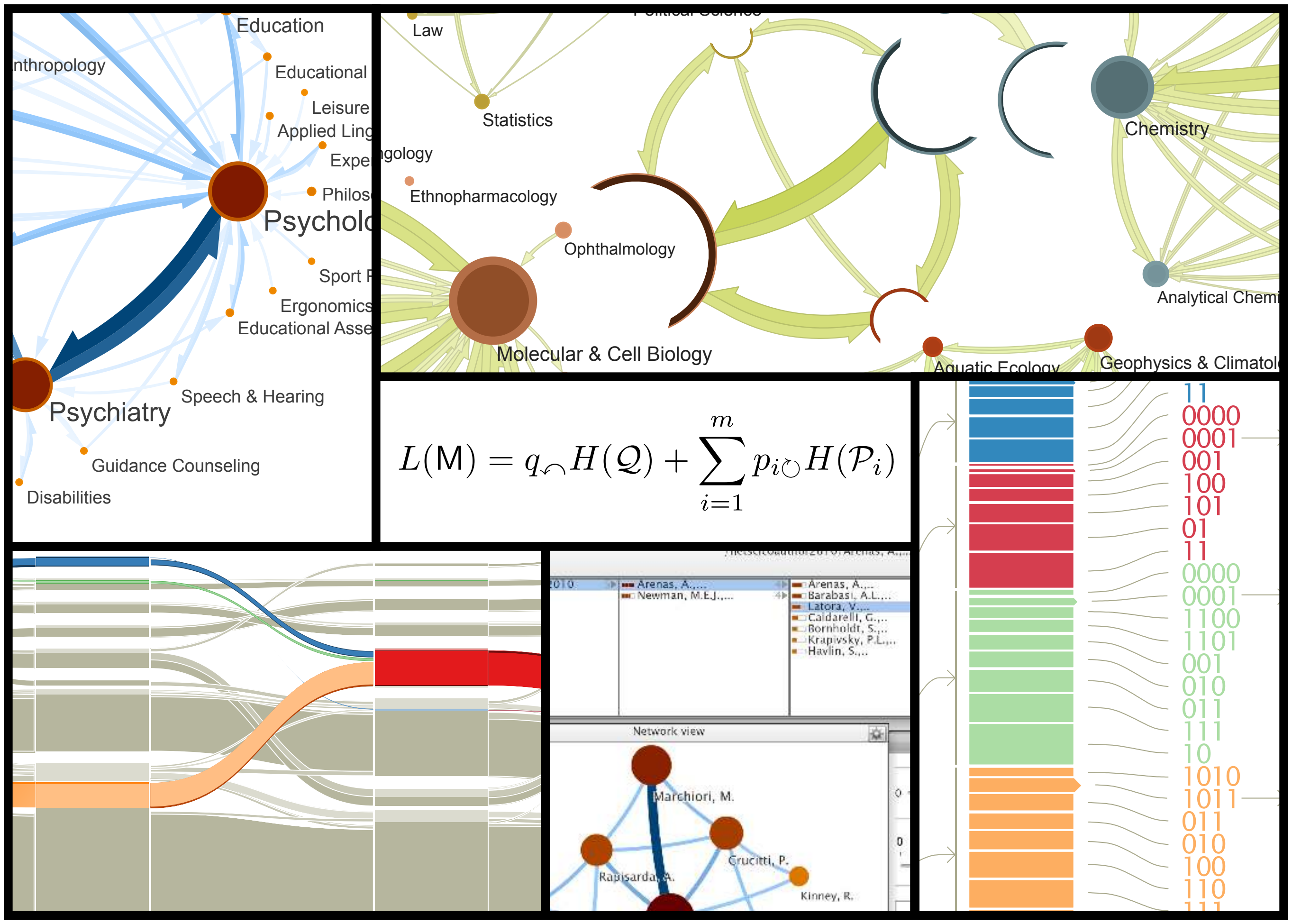


Figure 1 | The map equation framework consists of several tools for analyzing and visualizing large networks.

[*]Electronic address: martin.rosvall@physics.umu.se

Introduction

Ever since Aristotle put the basis of natural taxonomy, classification and categorization have played a central role in philosophical and scientific investigation to organize knowledge. To keep pace with the large amount of information that we collect in the sciences, scholars have explored different ways to automatically categorize data ever since the dawn of computer and information science.

We now live in the era of Big Data and fortunately we have several tools for classifying data from many different sources, including point clouds, images, text documents (1–3), and networks. Networks of nodes and links are simple yet powerful representations of datasets of interactions from a great number of different sources (4–6). Metabolic pathways, protein-protein interactions, gene regulation, food webs, the Internet, the World Wide Web, social interactions, and scientific collaboration are just a few examples of networked systems studied across the sciences.

In this tutorial, we will focus on classification of nodes into communities and on visualization of the community structure. In networks, communities refer to groups of nodes that are densely connected internally. Community detection in networks is challenging, and many algorithms have been proposed in the last few years to tackle this difficult problem. We will briefly mention some of the possible approaches to community detection in the next section. The rest of this chapter is devoted to providing theoretical background to the popular and efficient map equation framework and practical guidelines for how to use its accompanying search algorithm Infomap (7).

The current implementation of Infomap is both fast and accurate. It can classify millions of nodes in minutes and performs very well on synthetic data with planted communities (8, 9). Furthermore, the map equation framework is also naturally flexible and can be straightforwardly generalized to analyze different kinds of network data. For instance, Infomap not only provides two-level, multi-level, and overlapping solutions for analyzing undirected, directed, unweighted, and weighted networks, but also for analyzing networks that contain higher-order data, such as memory networks (10) and multiplex networks.

This tutorial is organized as follows. In Sec. 1, we provide some background on community detection in networks, in Sec. 2, we introduce the mathematics of the map equation and the Infomap algorithm, and, in Sec. 3, we explain how to run the software in the web applications and from the command line. We provide a collaboration network and a journal citation network as examples. For illustration, Fig. 1 shows a number of visualizations that can be created with the applications.

1. Overview of methods

Most networks display a highly organized structure. For instance, many social systems show *homophily* in their network representations: nodes with similar properties tend to form highly connected groups called communities, clusters, or modules. For a limited number of systems, we might have some information about the classification of the nodes. For example, Wikipedia is a large network of articles connected with hyperlinks, and each article is required to belong to at least one category. In general, however, these communities are not known a priori, and, in the few fortunate cases for which some information about the classification is available, it is often informative to integrate it with the information contained in the network structure. Therefore, community detection is one of the most used techniques among researchers when studying networks. Moreover, recommendation systems and network visualizations are just two of many highly useful applications of community detection.

A comprehensive overview of community-detection methods can be found in ref.(11). Here we just mention three of the main approaches:

Null models Methods based on null models compare some measure of connectivity within groups of nodes with the expected value in a proper null model (12–14). Communities are identified as the sets of nodes for which the connectivity deviates the most from the null model. This is the approach of modularity (12), which the commonly used Louvain method (13) implements.

Block models Methods based on block models identify blocks of nodes (15–17) with common properties. Nodes assigned to the same block are statistically equivalent in terms of their connectivity to nodes within the block and to other blocks. The latent block structure is identified by maximizing the likelihood of observing the empirical data.

Flow models Methods based on flows (7, 18) operate on the dynamics on the network rather than on its topological structure per se. The rationale is that the prime function of networks is to capture the flow between the components of the real systems they represent. Accordingly, communities consist of nodes among which flow persists for a long time once entered. As we explain in great detail in the next section, the map equation is a flow-based method.

The methods' performance can be evaluated using synthetic data generated with planted community structure, which the algorithms are supposed to detect from the network topology only (19). By performing these benchmark tests, it has been found that modularity optimization suffers from a so called resolution limit (20). A method is considered to have a resolution limit if the size of identified groups depends on the size of the whole network. A consequence is that well-defined modules can be merged together in large nerworks. Several solutions to this problem have been proposed. The Louvain method (13) provides a hierarchical tree of clusters based on local modularity maxima found during the optimization procedure. Another approach is to use so called resolution free methods with a tunable resolution parameter (21, 22).

One advantage of using the map equation framework is that the resolution limit of its two-level formulation depends on the total weight of links between communities rather than on the total weight of all links in the whole network (23). As a result, the outcome of the map equation is much less likely than modularity maximization to be affected by the resolution limit, without resorting to local maxima or tunable parameters. Moreover, for many networks the resolution limit vanishes completely in the multilevel formulation of the map equation.

Hierarchical block models (17) are also less likely to be affected by the resolution limit, and which method to pick ultimately depends on the particular system under study. For example, block models can also detect bipartite group structures, and are therefore well suited for analyzing food webs. On the other hand, the map equation framework is developed to capture how flow spreads across a system, and is therefore well suited for analyzing weighted and directed networks that represent the constraints on flow in the system. Since citation networks are inherently directed, the map equation framework is a natural choice for analyzing bibliometric networks.

Most of the algorithms mentioned above are implemented in open source software. Of particular relevance are three network libraries that provide several of these methods in a unified framework: NetworkX, graph-tool, and igraph. Although igraph also has a basic implementation of Infomap, we recommend the most updated and feature-rich implementation introduced in the next section and available from www.mapequation.org.

2. The map equation framework

Here we provide an overview of the map equation framework and the software for network analysis that we have developed on top of it and made available on www.mapequation.org. First we explain the flow-based and information-theoretic rationale behind the mathematics of the map equation, and then we outline the algorithm implemented in Infomap for minimizing the map equation over possible network partitions.

2.1. The map equation

The map equation is built on a flow-based and information-theoretic foundation and was first introduced in ref. (7). Specifically, the map equation takes advantage of the duality between finding community structure in networks and minimizing the description length of a random walker's movements on a network. That is, for a given modular partition of the network, there is an associated information cost for describing the movements of the random walker, or of empirical flow if available. Some partitions give shorter and some give longer description lengths. The partition with the shortest description length is the one that best captures the community structure of the network with respect to the dynamics on the network.

The underlying code structure of the map equation is designed such that the description can be compressed if the network has regions in which the random walker tends to stay for a long time. Therefore, with a random walker as a proxy for real flow, minimizing the map equation over all possible network partitions reveals important aspects of network structure with respect to the dynamics on the network. That is, the map equation is a direct measure of how well a given network partition captures modular regularities in the network.

The flow-based foundation of the map equation is well-suited for bibliometric analysis, since a random walker on a citation network is a good model for how researchers navigate scholarly literature, reading articles and following citations. See Fig. 2 for an illustration of a random walker moving across a network. For example, with a network of citing articles, or citing articles aggregated at the journal level, the modules identified by the map equation would correspond to research fields. Similarly, with a network of collaborators obtained from co-authorships, the modules identified by the map equation would correspond to research groups.

To explain the machinery of the map equation, we first derive its general expression and then illustrate with examples from the interactive map equation demo available on www.mapequation.org/apps/MapDemo.html (Figs. 2–4). However, we begin with a brief review of the foundations of the map equation: the mathematics of random walkers on networks and basic information theory. Readers not interested in the details of the map equation's theoretical foundation can skip to Sec. 2.1.4 for illustrative examples.

2.1.1. Dynamics on networks modeled with random walkers

The map equation measures the per-step theoretical lower limit of a modular description of a random walker on a network. Instead of measuring the codelength of a long walk and dividing by the number of steps, it is more efficient to derive the codelength from the stationary distribution of the random walker on the nodes and links. In general, given a network with n nodes and weighted directed links $W_{\alpha\to\beta}$ between nodes $\alpha,\beta \in 1,2,\ldots,n$, the conditional probability that the random walker steps from node α to node β is given by the relative link weight

$$p_{\alpha\to\beta} = \frac{W_{\alpha\to\beta}}{\sum_\beta W_{\alpha\to\beta}}. \tag{1}$$

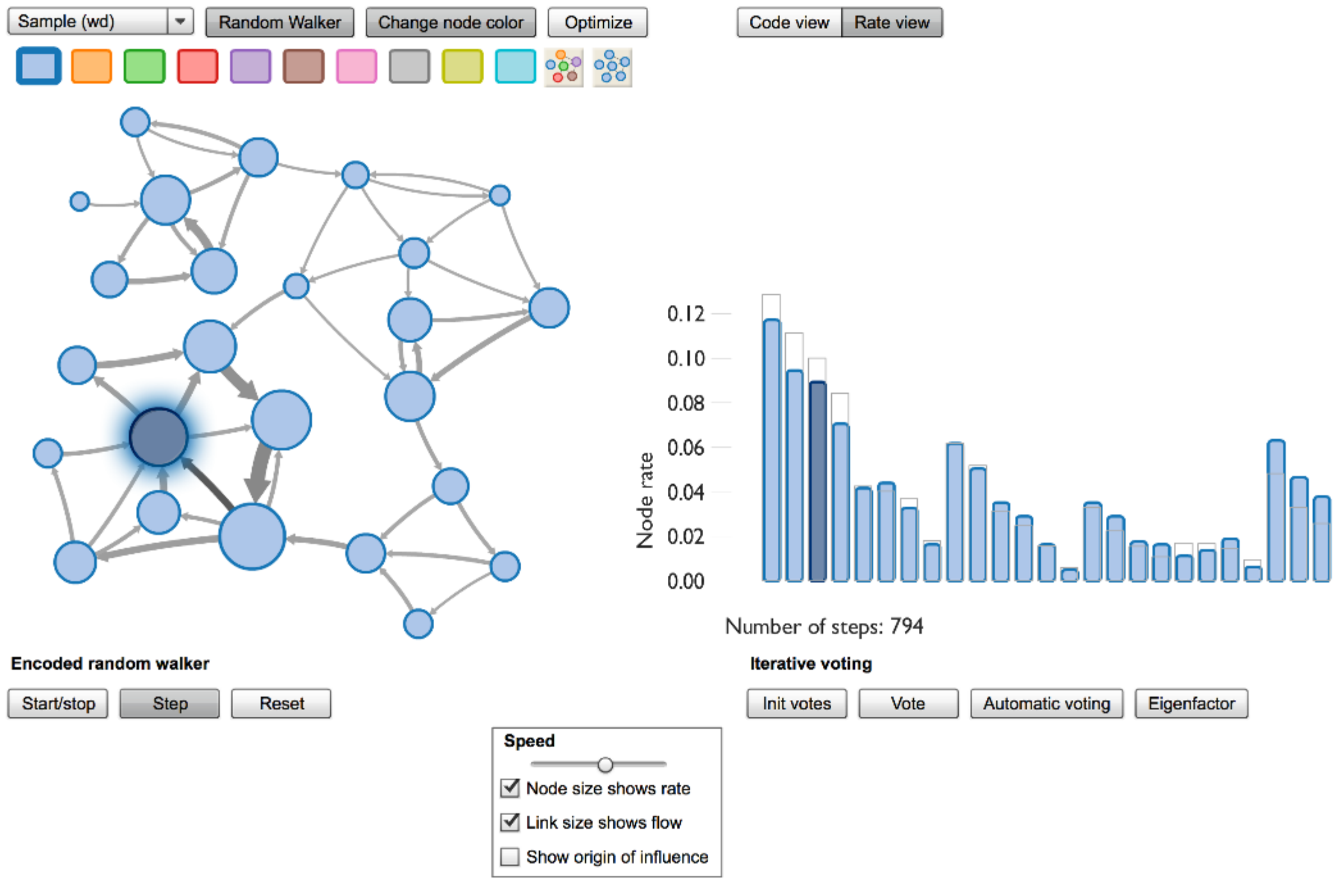


Figure 2 | The rate view of the map equation demo showing a random walker moving across the network. In the Flash application available on www.mapequation.org/apps/MapDemo.html, the random walker moves from node to node following the directed links proportional to their weights. In this snapshot, the random walker has taken 794 steps and visited the nodes according to the frequency distribution to the right. The currently visited node is highlighted both in the network and in the histogram.

Assuming that the stationary distribution is given by p_α, it can in principle be derived from the recursive system of equations

$$p_\alpha = \sum_\beta p_\beta p_{\beta\to\alpha}. \tag{2}$$

However, to ensure a unique solution independent of where the random walker starts in directed networks, at a low rate τ the random walker instead *teleports* to a random node. For more robust results that depend less on the teleportation parameter τ, we most often use teleportation to a node proportional to the total weights of the links to the node (24). The stationary distribution is then given by

$$p_\alpha = (1-\tau)\sum_\beta p_\beta p_{\beta\to\alpha} + \tau\frac{\sum_\beta W_{\beta\to\alpha}}{\sum_{\alpha,\beta} W_{\beta\to\alpha}}. \tag{3}$$

This system of equations can efficiently be solved with the power-iteration method (25).

To make the results even more robust to the teleportation parameter τ, we use *unrecorded* teleportation steps and only record steps along links (24). We capture these dynamics with an extra step without teleportation on a stationary solution similar to Eq. (3) but with teleportation to a node proportional to the total weights of the links *from* the node,

$$p^*_\alpha = (1-\tau)\sum_\beta p^*_\beta p_{\beta\to\alpha} + \tau\frac{\sum_\beta W_{\alpha\to\beta}}{\sum_{\alpha,\beta} W_{\beta\to\alpha}}. \tag{4}$$

The unrecorded visit rates on links $q_{\beta\to\alpha}$ and nodes p_α can now be expressed:

$$q_{\beta\to\alpha} = p^*_\beta p_{\beta\to\alpha} \tag{5}$$

$$p_\alpha = \sum_\beta q_{\beta\to\alpha}. \tag{6}$$

This so called smart teleportation scheme ensures that the solution is independent of where the random walker starts in directed networks with minimal impact on the results from the teleportation parameter. A typical value of the teleportation rate is $\tau = 0.15$, but in practice the clustering results show only small changes for teleportation rates in the range $\tau \in (0.05, 0.95)$ (24). For example, for undirected networks the results are completely independent of the teleportation rate and identical to results given by Eq. (2). For directed networks, a teleportation rate too close to 0 gives results that depend on how the random walker was initiated and should be avoided, but a teleportation value equal to 1 corresponds to using the link weights as the stationary distribution. Accordingly, the unrecorded teleportation scheme also makes it possible to describe raw flow given by the links themselves without first inducing dynamics with a random walker. The Infomap code described in Sec. 2.2 can use any of these dynamics described above, but we recommend the unrecorded teleportation scheme proportional to link weights for most robust results.

The map equation is free from external resolution parameters. Instead the resolution scale is set by the dynamics. The dynamics described above correspond to encoding one node visit per step of the random walker, but the code rate can be set both higher and lower (26). A higher code rate can be achieved by adding self-links and a lower code rate can be achieved by adding non-local links to the network (26). A higher code rate gives smaller modules because the random walker becomes trapped in smaller regions for a longer time. The Infomap code allows to increase the code rate from the natural value of encoding one node visit per step of the random walker.

2.1.2. Basic information theory

While the map equation gives the theoretical lower limit of a modular description of a random walker on a network, the interactive map equation demo illustrates the description with real codewords. We use Huffman codes (27), which are optimal in the sense that no binary codes can come closer to the theoretical limit. However, for identifying the optimal partition of the network, we are only interested in the compression rate and not the actual codewords. Accordingly, the Infomap algorithm only measures the theoretical limit given by the map equation.

Shannon's source coding theorem (28) states that the per step theoretical lower limit of describing a stream of n independent and identically-distributed random variables is given by the entropy of the probability distribution. That is, given the probability distribution $\mathcal{P} = \{p_i\}$ such that $\sum_i p_i = 1$, the lower limit of the per-step codelength is given by

$$L(\mathcal{P}) = H(\mathcal{P}) \equiv -\sum_i p_i \log p_i, \tag{7}$$

with the logarithm taken in base 2 to measure the codelength in bits. In other words, no codebook with codewords for the events distributed according to $\mathcal{P}$ can use fewer bits on average.

Accordingly, the best compression of random walker dynamics on a network is given by the entropy rate (28)

$$\sum_\alpha p_\alpha H(p_{\alpha\to\beta}), \tag{8}$$

which corresponds to the average codelength of specifying the next node visit given current node position, averaged over all node positions. This coding scheme takes advantage of the independent and identically distributed next node visits given current node position, but can not be used to take advantage of the modular structure of the network. Instead, the map equation uses the extra constraint that the only available information from one step to the next is the currently visited module, or that the random walk switches between modules, forcing independent and identically distributed events within and between modules. From this assumption naturally follows a modular description that is maximally compressed by the network partition that best represents the modular structure of the network with respect to the dynamics on the network.

2.1.3. The mathematics of the map equation

Given a network partition, the map equation specifies the theoretical modular description length of how concisely we can describe the trajectory of a random walker guided by the possibly weighted, directed links of the network. We use M to denote a network partition of the network's n nodes into m modules, with each node α assigned to a module i. We then seek to minimize the description length $L(\mathsf{M})$ given by the the map equation over possible network partitions M. Again, network partition that gives the shortest description length best captures the community structure of the network with respect to the dynamics on the network.

The map equation can be expressed in closed form by invoking Shannon's source coding theorem in Eq. (7) for each of multiple codebooks, and by weighting them by their rate of use. Both the description length and the rate of use can be expressed in terms of the node-visit rates p_α and the module-transition rates $q_{i\curvearrowleft}$ and $q_{i\curvearrowright}$ at which the random walker enters and exits each module i, respectively:

$$q_{i\curvearrowleft} = \sum_{\alpha \in j \neq i, \beta \in i} q_{\alpha \to \beta} \tag{9}$$

$$q_{i\curvearrowright} = \sum_{\alpha \in i, \beta \in j \neq i} q_{\alpha \to \beta}. \tag{10}$$

To take advantage of the modular structure of the network, m *module codebooks* and one *index codebook* are used to describe the random walker's movements within and between modules, respectively. Module codebook i has one codeword for each node $\alpha \in i$ and one exit codeword. The codeword lengths are derived from the frequencies at which the random walker visits each of the nodes in the module, $p_{\alpha \in i}$, and exits the module, $q_{i\curvearrowright}$. We use $p_{i\circlearrowright}$ to denote the sum of these frequencies, the total use of codewords in module i, and $\mathcal{P}^i$ to denote the normalized probability distribution. Similarly, the index codebook has codewords for module entries. The codeword lengths are derived from the set of frequencies at which the random walker enters each module, $q_{i\curvearrowleft}$. We use $q_{\curvearrowleft}$ to denote the sum of these frequencies, the total use of codewords to move into modules, and Q to denote the normalized probability distribution. We want to express average length of codewords from the index codebook and the module codebooks weighted by their rates of use. Therefore, the map equation is

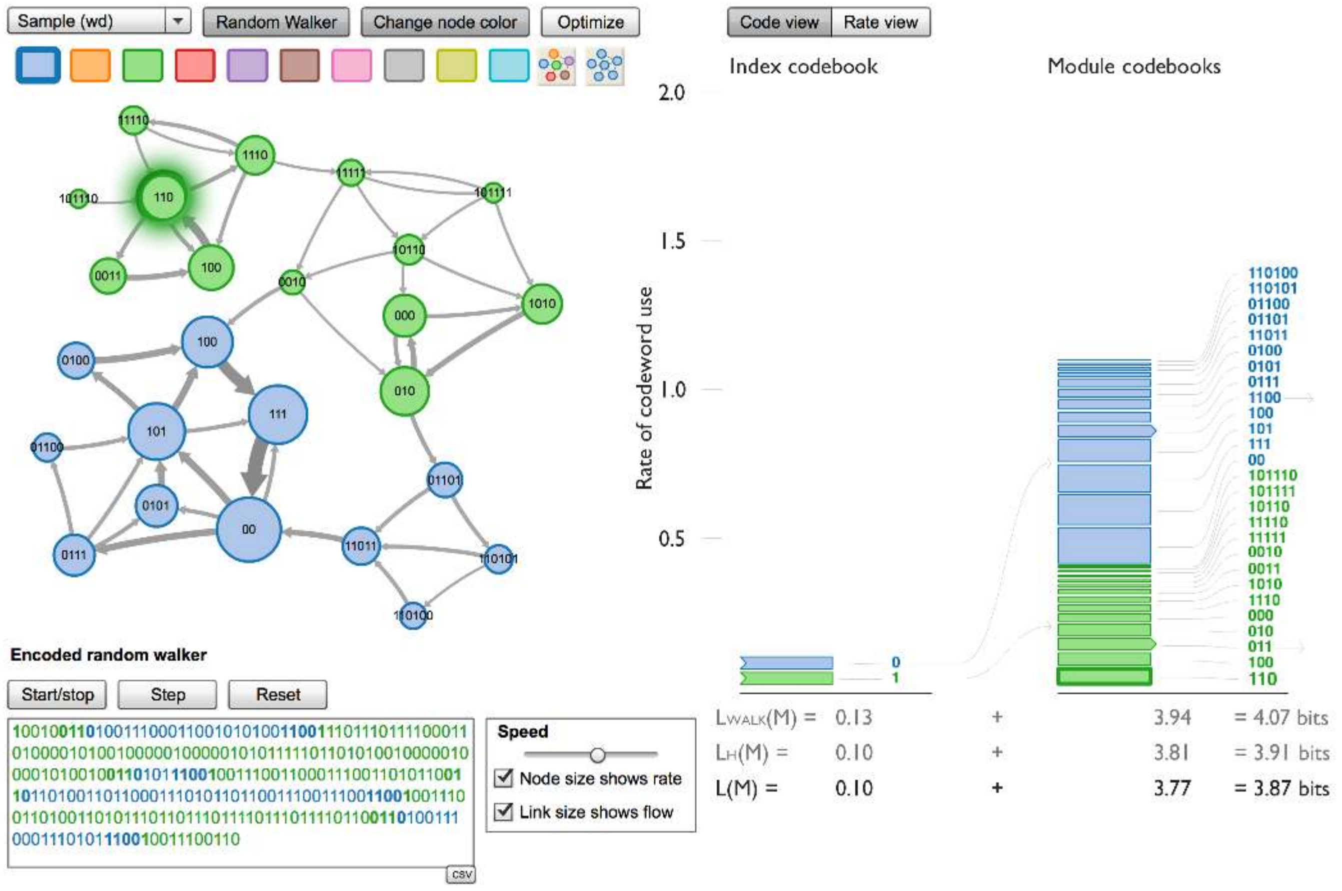
$$L(\mathsf{M}) = q_{\curvearrowleft} H(Q) + \sum_{i=1}^{m} p_{i\circlearrowright} H(\mathcal{P}_i). \tag{11}$$

Below we explain the terms of the map equation in detail and in Figs. 3 and 4 we provide examples with Huffman codes for illustration.

$L(\mathsf{M})$	The per-step description length for module partition M. That is, for module partition M of n nodes into m modules, the lower bound of the average length of the code describing a step of the random walker. The bottom right bit counts in Figs. 3 and 4 show the description length for the given network partition.
$q_{\curvearrowleft} = \sum_{i=1}^{m} q_{i\curvearrowleft}$	The rate at which the index codebook is used. The per-step use rate of the index codebook is given by the total probability that the random walker enters any of the m modules. The total height of the blocks under *Index codebook* in Figs. 3 and 4 corresponds to this rate.
$H(Q) = -\sum_{i=1}^{m} (q_{i\curvearrowleft}/q_{\curvearrowleft}) \log(q_{i\curvearrowleft}/q_{\curvearrowleft})$	The frequency-weighted average length of codewords in the index codebook. The entropy of the relative rates to use the module codebooks measures the smallest average codeword length that is theoretically possible. The heights of individual blocks under *Index codebook* in Figs. 3 and 4 correspond to the relative rates and the codeword lengths approximately correspond to the negative logarithm of the rates in base 2.
$p_{i\circlearrowright} = \sum_{\alpha \in i} p_\alpha + q_{i\curvearrowright}$	The rate at which the module codebook i is used, which is given by the total probability that any node in the module is visited, plus the probability that the random walker exits the module and the exit codeword is used. This rate corresponds to the total height of similarly colored blocks associated with the same module under *Module codebooks* in Figs. 3 and 4.
$H(\mathcal{P}^i) = -(q_{i\curvearrowright}/p_{i\circlearrowright}) \log(q_{i\curvearrowright}/p_{i\circlearrowright}) - \sum_{\alpha \in i} (p_\alpha/p_{i\circlearrowright}) \log(p_\alpha/p_{i\circlearrowright})$	The frequency-weighted average length of codewords in module codebook i. The entropy of the relative rates at which the random walker exits module i and visits each node in module i measures the smallest average codeword length that is theoretically possible. The heights of individual blocks under *Module codebooks* in Figs. 3 and 4 correspond to the relative rates and the codeword lengths approximately correspond to the negative logarithm of the rates in base 2.

2.1.4. The machinery of the map equation

The map equation demo available on www.mapequation.org/apps/MapDemo.html provides an interactive interface to help understand the machinery of the map equation. The demo has two modes accessible with the two buttons $\boxed{\text{Rate view}}$ and $\boxed{\text{Code view}}$. The purpose of the rate view shown in Fig. 2 is to illustrate how we use the random walker to induce flow on



```
10010011010011100011001010100110011101110111100011
0100001010010000010000010101111101101010010000010
0001010010011010111001001110011000111001101011001
10110100110110001110101101100111001110011001001110
01101001101011101101110111101110111101100110100111
0001110101110010011100110
```

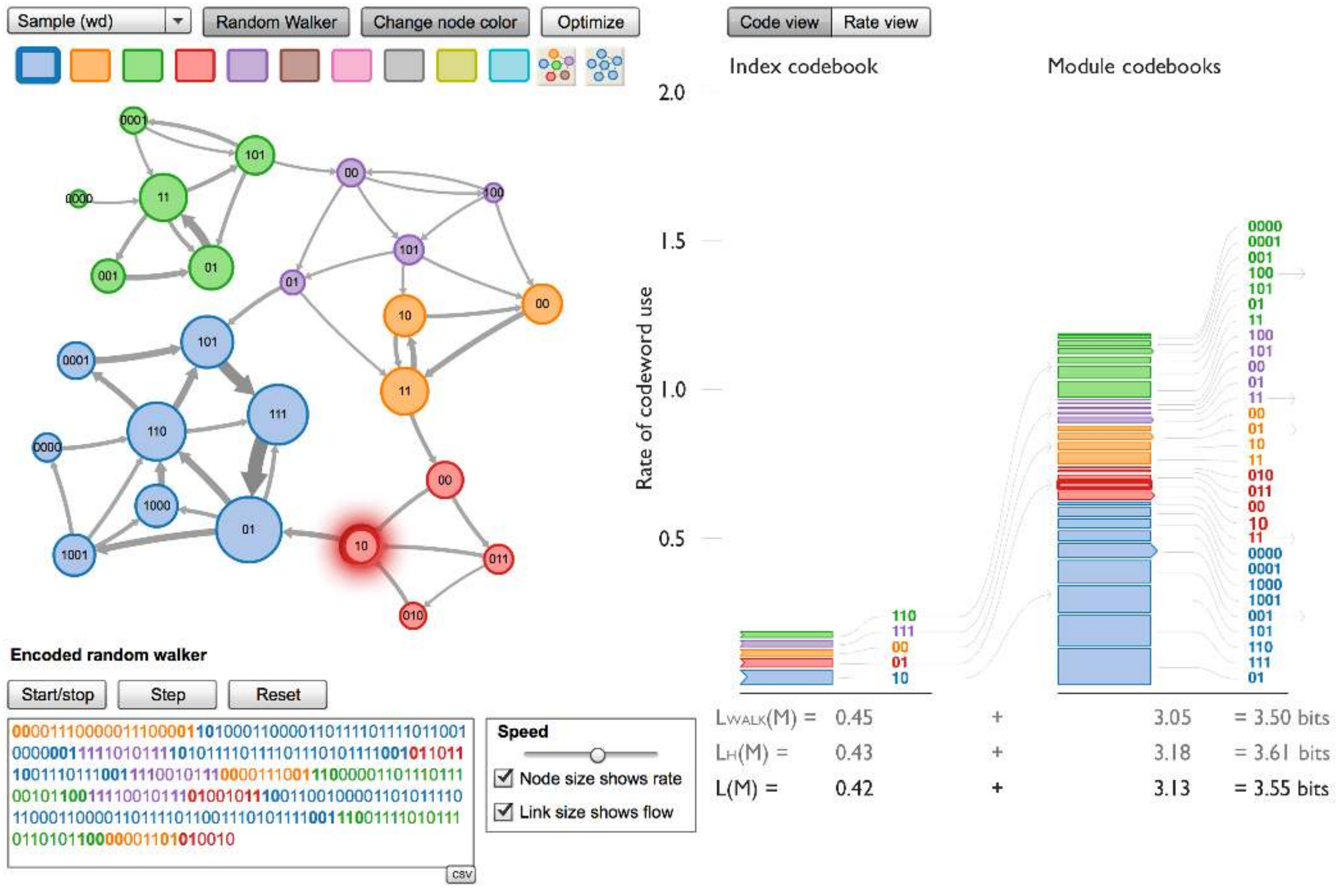
```
LWALK(M) = 0.13 + 3.94 = 4.07 bits
LH(M) =    0.10 + 3.81 = 3.91 bits
L(M) =     0.10 + 3.77 = 3.87 bits
```

Figure 3 | The code view of the map equation demo showing the encoding of a random walker with a two-module solution. In the Flash application available on www.mapequation.org/apps/MapDemo.html, the random walker currently visits the green node with codeword 110. The height of a block under *Index codebook* represents the rate at which the random walker enters the module. The bit sequence next to each block is the associated codeword. Similarly, the height of a block under *Module codebooks* represents the rate at which the random walker visits a node, or exits a module. The blocks representing exit rates have a arrow on their right side. The text field in the bottom left corner shows the encoding for the previous steps of the random walker, ending with the step on the node with codeword 110. Steps in the two modules are highlighted with green and blue, respectively, and the enter and exit codewords are boldfaced. $L(\mathsf{M})$ in the bottom right corner shows the theoretical limit of the description length for the chosen two-module network partition given by Eq. 11. $L_{\mathsf{M}}(\mathsf{M})$ shows the limit of the Huffman coding given by the actual codebooks and $L_{\mathsf{WALK}}(\mathsf{M})$ shows the average per-step description length of the realized walk in the simulation.

the network from the constraints set by the link structure. The purpose of the code view shown in Figs. 3 and 4 is to illustrate the duality between finding regularities in the network structure and compressing a description of the flow induced by the network structure.

In the rate view, click $\boxed{\text{Random walker}}$ and $\boxed{\text{Start/stop}}$ and a random walker begins traversing the network. Moving from one node to another, the random walker chooses which neighbor to move to next proportional to the weights of the links to the neighbors according to Eq. (1). As described in Sec. 2.1.1, to ensure an ergodic solution, i.e., that the average visit rates will reach a steady-state solution independent of where the random walker starts, the random walker sometimes moves to a random node irrespective of the link structure. In this implementation, the random walker *teleports* to a random node with 15 percent chance every step, or about every six steps.

The histogram on the right in the rate view shows the node-visit distribution. The colored bars show the average distribution so far in the simulation, and the bars with a gray border show the ergodic solution. The visit rates of the ergodic solution correspond to the eigenvector of the leading eigenvalue of the transition matrix given by the network. This solution also corresponds to the PageRank of the nodes (29). After a long time the average visit rates of the random walker will approach the ergodic solution, but this walk-based method is very inefficient for obtaining the ergodic solution. In practice, since the map equation only takes the ergodic visit rates as input, we use the power-iteration method to derive the ergodic solution (25). The power-iteration method works by operating on the probability distribution of random walkers rather than on a specific random walker. By pressing $\boxed{\text{Init votes}}$, each node receives an equal share of this probability. By clicking $\boxed{\text{Vote}}$, the probability at each node is pushed to neighbors proportional to the link weights, and 15 percent is distributed randomly. As



☑ Node size shows rate
☑ Link size shows flow

```
00001110000011100001101000110000110111101111011001
00000011111010111101011110111101110101111001011011
10011101110011110010111000011100111000001101110111
0010110011110010111010010111001100100001101011110
11000110000110111101100111010111100111001111010111
011010110000001101010010
```

L				

Figure 4 | **The code view of the map equation demo showing the encoding of a random walker with the optimal five-module solution.** Compared to the two-module solution in Fig. 3, the index codebook is larger and used more often. Nevertheless, and thanks to the more efficient encoding of movements within modules with the smaller module codebooks, the per-step codelength is 0.32 bits shorter on average.

can be seen by clicking a few times on $\boxed{\text{Vote}}$, the probability distribution quickly approaches the ergodic solution.

In the code view, each node is labeled with its codeword as shown in Figs. 3 and 4. Each event, i.e., that the random walker visits a node, enters a module, or exits a module, is also represented as a block in the stacks on the right. The stack under *Index codebook* shows module-enter events, and the stack under *Module codebooks* shows within-module events. Mouseover a node or a block in the map equation demo highlights the corresponding block or node. The height of a block represents the rate at which the corresponding event occurs, and the bit string to the right of each block is the codeword associated with the event. The codewords are Huffman codes (27) derived from their frequency of use. Huffman codes are optimal for symbol-by-symbol encoding with binary codewords given a known probability distribution. As explained in Sec. 2.1.2, the average codelength of a Huffman code is bounded below by the entropy of the probability distribution (28). In general, the average codelength of a Huffman code is somewhat longer than the theoretical limit given by the entropy, which has no constraints on using integer-length codewords. For example, the average codelengths with actual binary codewords shown in the lower right corners of Figs. 3 and 4 are about a percent longer than the theoretical limit.

In practice, for taking advantage of the duality between finding the community structure and minimizing the description length, we use the theoretical limit given by the map equation. That is, we show the codewords in the map equation demo only for pedagogical reasons. For example, note that frequently visited nodes are assigned short codewords and that infrequently visited nodes are assigned longer codewords, such that the description length will be short on average. Similarly, modules which the random walker enters frequently are assigned short codewords, and so on. The varying codeword lengths take advantage of the regularity that some events happen more frequently than other, but does not take advantage of the community structure of the network. Instead, it is the modular code structure with an index codebook and module codebooks that exploits the community structure.

The optimal network partition corresponds to a modular code structure that balances the cost of specifying movements within and between modules. Figure 3 shows a network partition with two modules. The lower bound of the average description length for specifying movements within modules is 3.77 bits per step and only 0.10 bit per step for specifying movements into modules. Describing movements into modules is cheap because a single bit is necessary to specify which of the two modules the random walker enters once it switches between modules, and it only switches between modules every ten steps on average. However, the movements within modules are relatively expensive, since each module contains many nodes, each one with a rather long codeword. The more events a codebook contains, the longer must the codewords be on average, because there is a limited number of short codewords. Figure 4 shows the optimal network partition with five modules. In this partition, the smaller modules allow for more efficient encoding of movements within modules. On average, specifying movements within modules requires 3.13 bits per step, 0.64 bit less in the two-module solution in Fig. 3. This compression gain comes at the cost of more expensive description of movements into the five modules, but the overall codelength is nevertheless smaller, 3.55 bits in the optimal solution compared to 3.87 bits in the two-module solution. To better understand the inner-workings of the map equation, it is helpful to change the partition of the network in the map equation demo and study how the code structure and associated codelengths change.

Here we have described the basic two-level map equation. One strength of the map equation framework is that it is generalizable to higher-order structures, for example to hierarchical structures (30), overlapping structures (31), or higher-order Markov dynamics (10). For details about those methods, we refer to the cited papers.

2.2. Infomap

We use *Infomap* to refer to the search algorithm for minimizing the map equation over possible network partitions. Below we briefly describe the algorithms for identifying two-level and multilevel solutions.

2.2.1. Two-level algorithm

The core of the algorithm follows closely the Louvain method (13): neighboring nodes are joined into modules, which subsequently are joined into supermodules and so on. First, each node is assigned to its own module. Then, in random sequential order, each node is moved to the neighboring module that results in the largest decrease of the map equation. If no move results in a decrease of the map equation, the node stays in its original module. This procedure is repeated, each time in a new random sequential order, until no move generates a decrease of the map equation. Then the network is rebuilt, with the modules of the last level forming the nodes at this level, and, exactly as at the previous level, the nodes are joined into modules. This hierarchical rebuilding of the network is repeated until the map equation cannot be reduced further.

With this algorithm, a fairly good clustering of the network can be found in a very short time. Let us call this the core algorithm and see how it can be improved. The nodes assigned to the same module are forced to move jointly when the network is rebuilt. As a result, what was an optimal move early in the algorithm might have the opposite effect later in the algorithm. Two or more modules that merge together and form one single module when the network is rebuilt can never be separated again in this algorithm. Therefore, the accuracy can be improved by breaking the modules of the final state of the core algorithm in either of the two following ways:

Submodule movements. First, each cluster is treated as a network on its own and the main algorithm is applied to this network. This procedure generates one or more submodules for each module. Then all submodules are moved back to their respective modules of the previous step. At this stage, with the same partition as in the previous step but with each submodule being freely movable between the modules, the main algorithm is re-applied.

Single-node movements. First, each node is re-assigned to be the sole member of its own module, in order to allow for single-node movements. Then all nodes are moved back to their respective modules of the previous step. At this stage, with the same partition as in the previous step but with each single node being freely movable between the modules, the main algorithm is re-applied. In practice, we repeat the two extensions to the core algorithm in sequence and as long as the clustering is improved. Moreover, we apply the submodule movements recursively. That is, to find the submodules to be moved, the algorithm first splits the submodules into subsubmodules, subsubsubmodules, and so on until no further splits are possible. Finally, because the algorithm is stochastic and fast, we can restart the algorithm from scratch every time the clustering cannot be improved further and the algorithm stops. The implementation is straightforward and, by repeating the search more than once, 100 times or more if possible, the final partition is less likely to correspond to a local minimum. For each iteration, we record the clustering if the description length is shorter than the shortest description length recorded before.

2.2.2. Multilevel algorithm

We have generalized our search algorithm for the two-level map equation to recursively search for multilevel solutions. The recursive search operates on a module at any level; this can be all the nodes in the entire network, or a few nodes at the finest level. For a given module, the algorithm first generates submodules if this gives a shorter description length. If not, the recursive search does not go further down this branch. But if adding submodules gives a shorter description length, the algorithm tests if movements within the module can be further compressed by additional index codebooks. Further compression can be achieved both by adding one or more coarser codebooks to compress movements between submodules or by adding one or more finer index codebooks to compress movements within submodules. To test for all combinations, the algorithm calls itself recursively, both operating on the network formed by the submodules and on the networks formed by the nodes within every submodule. In this way, the algorithm successively increases and decreases the depth of different branches of the multilevel code structure in its search for the optimal hierarchical partitioning. For every split of a module into submodules, we use the two-level search algorithm described above.

3. Step-by-step instructions to the MapEquation software package

Here we provide detailed instructions on how to analyze networks with the map equation framework and the software we have developed on top of it. Networks can be analyzed either by using the MapEquation web applications, or by downloading the Infomap source code and run the program locally from the command line. Both the MapEquation software package and the Infomap source code can be found on the website www.mapequation.org. The MapEquation software package provides applications for both analyzing and visualizing networks. The web application interface offers helpful tools for visualizing the data, but the interface can be slow for large networks and is not useful for clustering more than on the order of 10,000 nodes. The Infomap source code, on the other hand, is fast and more flexible. It can cluster networks with hundreds of millions of nodes and comes with additional options for network analysis. After using Infomap, many of the results can be imported and visualized in the web application.

This section is organized as follows. In Sec. 3.1, we provide instructions for how to analyze and visualize networks with the MapEquation web applications. In Sec. 3.2, we explain how to analyze networks with the Infomap source code from the command line. Finally, in Sec. 3.3, we specify all available input and output formats.

3.1. The MapEquation web applications

The MapEquation web applications are available on www.mapequation.org/apps.html. They include the *Map & Alluvial Generator* for visualizing two-level modular structures and change in those structures, and the *Hierarchical Network Navigator* for exploring hierarchical and modular organization of real-world networks. Below we provide step-by-step instructions for how to use these applications.

3.1.1. The Map Generator

The Flash application on www.mapequation.org/apps/MapGenerator.html includes the Map Generator and the Alluvial Generator for analyzing and visualizing single networks and change in multiple networks, respectively. When you load your weighted or unweighted, directed or undirected network into the application, the Map Generator clusters the network based on the map equation and generates a map for you to customize and save as a pdf file. To simplify and summarize important structural changes in networks with alluvial diagrams, the Alluvial Generator makes it possible to load multiple networks with coinciding node names and partition them one by one with the Map Generator. Figure 5 shows the start frame of the Map and Alluvial Generator for loading networks.

To load, analyze, and view networks with the Map Generator, the following steps are essential:

1. Load the `.net` network file into the Map Generator by clicking the button $\boxed{\text{Load network}}$ and choose between undirected and directed network. For very large networks, the load and clustering time can be long in the Flash-based Map Generator. If you are encountering problems because of large networks, you can run the clustering code offline and load the `.map` file into the application by clicking $\boxed{\text{Load map}}$.

   If you just want to try out the Map Generator, a few sample networks are provided in the application, including *Modular demo network*, *Network scientists 2010*, and *Social science 2004*. The *Modular demo network* is the network used in Sec. 2.1. The weighted undirected network *Network scientists 2010* is the largest connected component of a coauthorship network compiled (for details of how weights are assigned, see ref. (32)) from two network reviews, refs. (33) and (34) and one community detection review, ref. (11), and can be downloaded[1] in `.net` format. The weighted directed network *Social science 2004*, as well as *Science 1998–2004* provided under $\boxed{\text{Load map}}$, come from Thomson-Reuters' Journal Citation Reports 2004. The data tally on a journal-by-journal basis the citations from articles published in a given year to articles published in the previous five years, with self-citations excluded.

2. Cluster the network based on the map equation by clicking $\boxed{\text{Calculate clusters}}$ or alternatively provide a clustering in Pajek's `.clu` format by clicking $\boxed{\text{Load cluster data}}$. The Infomap algorithm tries to minimize the description length of a random walker's movement on the network as described in Sec. 2.2 and reveals important aspects of network structure with respect to the dynamics on the network.

[1] Collaboration network of network scientists is available for download here: http://mapequation.org/downloads/netscicoauthor2010.net

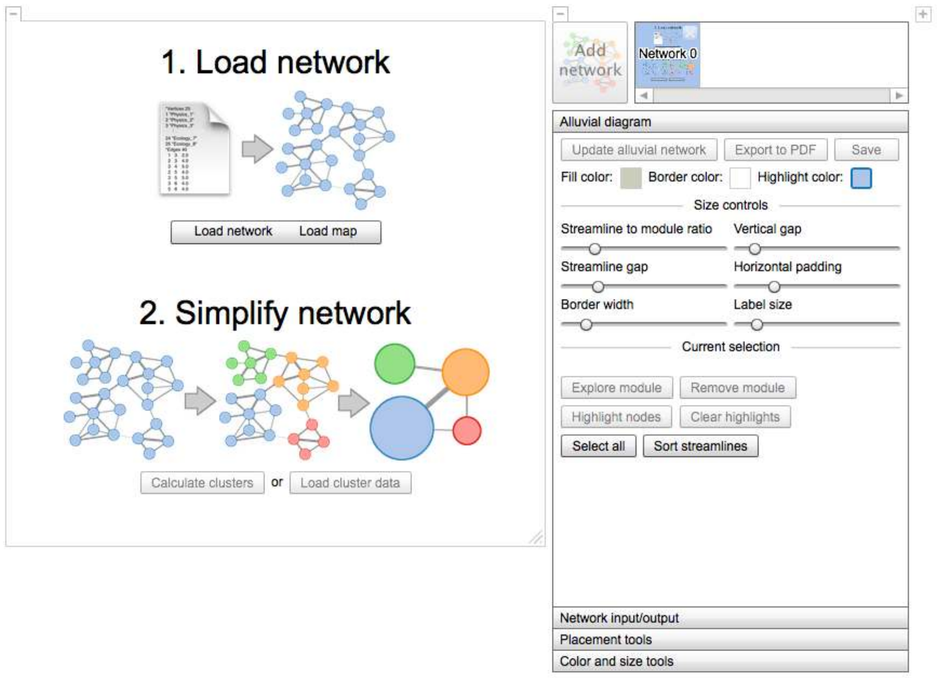
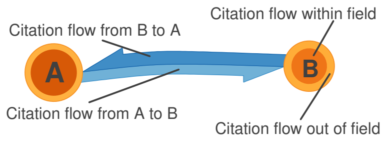


Figure 5 | The start frame of the Map and Alluvial Generator in the Flash application available on www.mapequation.org/apps/MapGenerator.html. Load the `.net` network file into the Map Generator by clicking the button $\boxed{\text{Load network}}$ and choose between undirected and directed network. When the network is loaded, click $\boxed{\text{Calculate clusters}}$ to cluster the network based on the map equation and generate a map of the network.

3. The Map Generator displays the network as a map. Every module represents a cluster of nodes and the links between the modules represent the flow between the modules. The default name of a module is given by the node with largest flow volume in the module. The size of a module is proportional to the average time a random walker spends on nodes in the module and the width of a link is proportional to the per step probability that a random walker moves between the modules. The module's internal flow is also distinguished from the flow out of the module by a layer as shown for citation data below.


Citation flow within field
Citation flow from B to A
A
B
Citation flow from A to B
Citation flow out of field

4. Customize the map by changing the position of the modules manually or automatically (see $\boxed{\text{Placement tools}}$ in the control panel), by changing the names of the modules (double click on a module to edit the name and list the nodes within the module together with their flow values), by changing the color of modules and links (see $\boxed{\text{Color and size tools}}$ in the control panel), by moving the labels, etc. All adjustments can also be applied only to selected modules (shift-click selects a single module and shift-drag selects multiple modules).

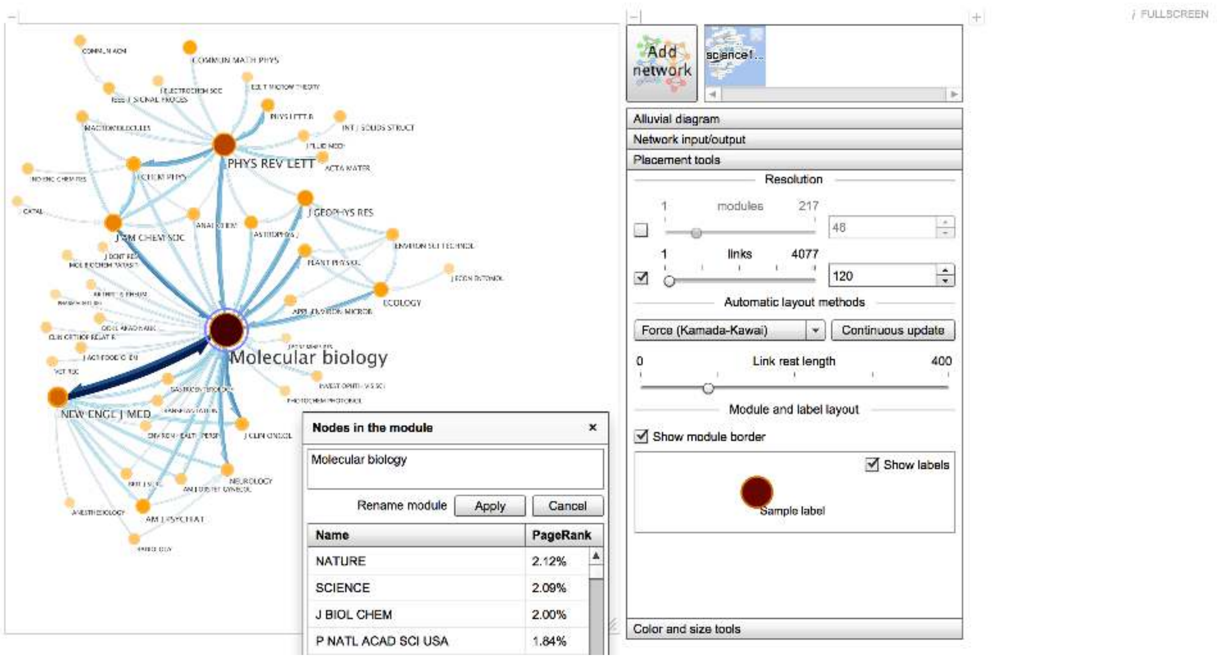
5. Save the customized map in scalable vector graphics as a pdf file or as a `.map` file for later access in the Map Generator.

Figure 6 shows the Map Generator after following steps 1-5 above with the journal citation network *Science 2004* provided in the application.

3.1.2. The Alluvial Generator

The Alluvial Generator is integrated with the Map Generator. It can load multiple networks with coinciding node names and partition them one by one with the Map Generator. The alluvial diagram can reveal organizational changes with streamlines between modules of the loaded networks, as shown on the right in Fig.7. The figure shows three scientific journal networks for years 1998, 2001, and 2004 loaded with the button $\boxed{\text{Load map}}$.

To use the Alluvial Generator, the following steps are essential:



Resolution

☐ 1 modules 217: 48

☑ 1 links 4077: 120

Automatic layout methods

Force (Kamada-Kawai) | Continuous update

0 Link rest length 400

Module and label layout

☑ Show module border

☑ Show labels

Sample label

Nodes in the module

Molecular biology

Rename module | Apply | Cancel

Name	PageRank
NATURE	2.12%
SCIENCE	2.09%
J BIOL CHEM	2.00%
P NATL ACAD SCI USA	1.84%

Figure 6 | The map and the control panel of the Map Generator available on www.mapequation.org/apps/MapGenerator.html. In this example, the network *Science 2004* was loaded and the scientific fields were identified and visualized by clicking $\boxed{\text{Calculate clusters}}$. The largest field has been renamed to *Molecular biology* and the open window lists the top journals assigned to the field together with their flow volume (PageRank).

1. Follow the instructions for the Map Generator described in Sec. 3.1.1 above to load a first network.

2. Click $\boxed{\text{Add network}}$ above the control panel to load additional networks. You can rearrange the order of loaded networks. Simply click a network thumbnail above the control panel and drag it to its preferred position.

3. The alluvial diagram is displayed to the right of the control panel. If you need more room, you can collapse the map by clicking the collapse button in the upper left corner of the map. It is easy to rearrange the modules and the streamlines, just click and drag to the new position. Highlighted nodes in a module can be rearranged if you keep the mouse button pressed for two seconds. The modules are first named by the most important node in the module (highest PageRank), but all names can be selected and changed appropriately. To change the layout of the diagram, use the size controls under $\boxed{\text{Alluvial diagram}}$ in the control panel.

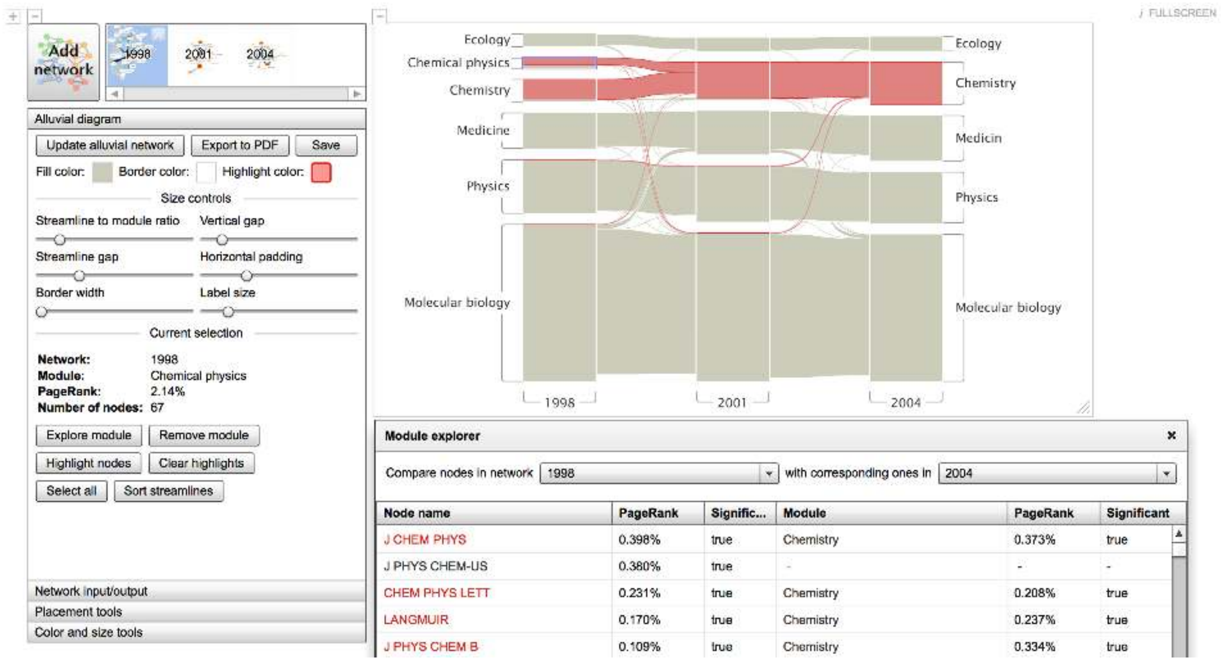
4. To remove, highlight, or explore a module, just click the module and select one of the options under $\boxed{\text{Alluvial diagram}}$ in the control panel. A double-click takes you directly to the *Module explorer*. Drag and select multiple modules to perform actions to multiple modules at the same time.

5. In the Module explorer, you can select and highlight individual or groups of nodes. The left column corresponds to the selected module(s) and the right column corresponds to the module assignment in the network marked in the drop-down list. Grayed out names belong to modules that are not included in the diagram. To include such a module, just double-click the grayed-out module name. A dash – means that the node does not exist in the network.

6. By clicking $\boxed{\text{FULLSCREEN}}$ in the upper right corner you can use your entire screen. For security reasons, Flash does not allow for inputs from the keyboard in fullscreen mode and you cannot edit any text. Pressing $\boxed{\text{ESC}}$ takes you back to normal mode.

To separate change from mere noise, we provide separate code that simultaneously identifies modules and performs significance analysis with bootstrap networks. The code outputs a file with extension `.smap` described in Sec. 3.3 below. To include significance information about the network clusters, download and run the code `conf-infomap`[2] on each network and load

[2] Code for generating significance modules is available for download here: www.tp.umu.se/ rosvall/code.html#mapchange



Network: 1998
Module: Chemical physics
PageRank: 2.14%
Number of nodes: 67

Compare nodes in network 1998 with corresponding ones in 2004

Node name	PageRank	Signific...	Module	PageRank	Significant
J CHEM PHYS	0.398%	true	Chemistry	0.373%	true
J PHYS CHEM-US	0.380%	true	-	-	-
CHEM PHYS LETT	0.231%	true	Chemistry	0.208%	true
LANGMUIR	0.170%	true	Chemistry	0.237%	true
J PHYS CHEM B	0.109%	true	Chemistry	0.334%	true

Figure 7 | The control panel and an alluvial diagram of the Alluvial Generator available on www.mapequation.org/apps/MapGenerator.html. In this example, the three journal citation networks *Science 1998*, *Science 2001*, and *Science 2004* have been loaded as `.map` files. The alluvial diagram uses streamlines to connect nodes assigned to modules in the different networks. Here all journals assigned to the field Chemistry in 2004 have been highlighted in red, and the streamlines trace back in which fields those journals were clustered in years 1998 and 2001. The Module explorer contains detailed information about individual journals.

the resulting `.smap` files instead of the networks by clicking $\boxed{\text{Load map}}$.

3.1.3. The Hierarchical Network Navigator

We have developed the Hierarchical Network Navigator to make it easier to explore the hierarchical and modular organization of real-world networks. When you load your network, the network navigator first runs the Infomap algorithm to generate a hierarchical map. Then it loads the solution into the *Finder* and *Network view* for you to explore. The Finder simultaneously shows modules in multiple levels but no link structure, whereas the Network view shows the link structure within a single module. Figure 8 shows the Hierarchical Network Navigator loaded with the undirected network *Network scientists 2010* provided in the application.

To use the Hierarchical Network Navigator, the following steps are essential:

1. Click on the button $\boxed{\text{Load network}}$ and browse your network file matching the file formats.
2. Choose between Undirected/Directed links and Multilevel/Two-level clustering.
3. If you loaded a link list, you have to check Zero-based numbering if the node numbers in the file starts from 0, otherwise they are assumed to start from 1.
4. Click on $\boxed{\text{Calculate clusters}}$.
5. The Infomap algorithm will start to cluster your network and print its progress in the Network loader window. When it is finished, the Network loader window will automatically close and you can navigate your network in the Finder and the Network view. If Infomap exited with an error, please check the error message in the Network loader window.

Because the Hierarchical Network Navigator makes it possible to navigate really large networks, we have developed a streamable file format that supports fast navigation without first loading the entire network file and subsequently clustering the network in the Flash application with the integrated and therefore slower Infomap code as described in steps 1–5 above. The binary file with suffix `.bftree` can be generated with the fast Infomap source code described in Sec. 3.2. The file format includes the hierarchical structure in a breath first order, including the flow links of each sub-network. In this way, only a small part of the file needs to be loaded to visualize the top structures, and the deeper structures can be loaded on demand. If the stand-alone Infomap source code is used, the following steps are essential:

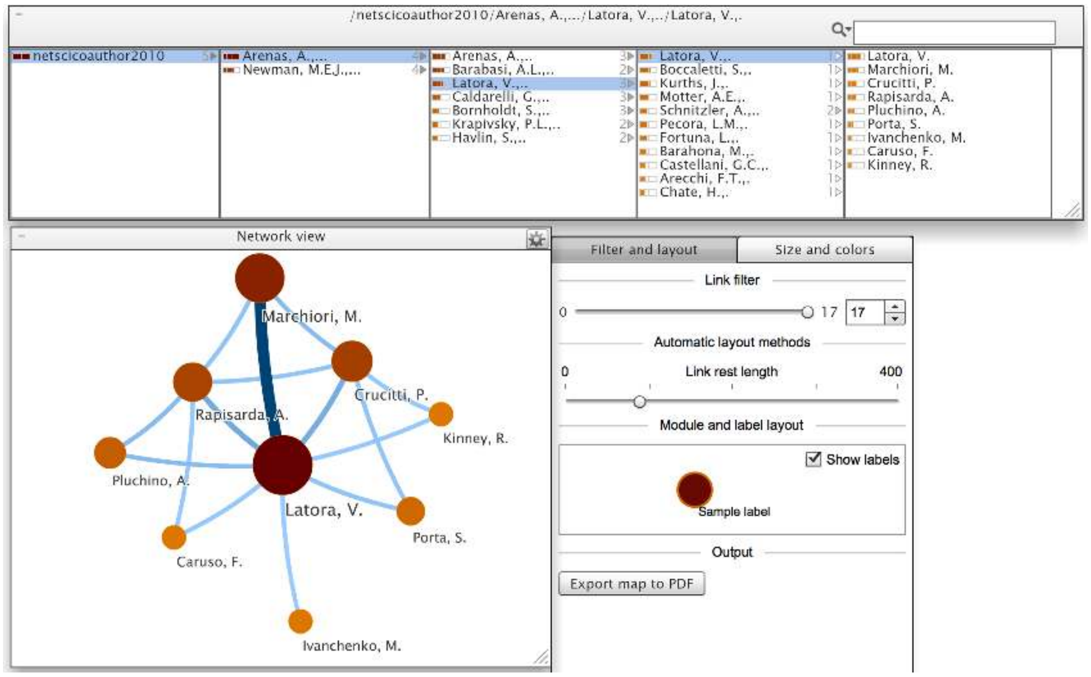


Figure 8 | The Finder and the Network view in the Hierarchical Network Navigator available on www.mapequation.org/apps/NetworkNavigator.html. In this example, we have loaded the collaboration network *Network scientists 2010* provided in the application and navigated three steps down to the finest level in the hierarchical organization to show the connections between the actual scientists in the module *Latora, V.,.* named after the researcher with strongest connections and largest flow value in the module. We use a period or multiple periods after a module name to indicate that there are one or more levels of nested modules within the module.

1. Run Infomap on the network with the flag **`--bftree`** (see instructions in Sec. 3.2).
2. Click $\boxed{\text{Load map}}$ in the Hierarchical Network Navigator and browse to your `.bftree` file generated by Infomap.
3. The Network loader window will automatically close and the Finder and Network view will open with the top level data visible.

Whichever method was used to generate the result, the following commands are available to navigate and customize the network:

Navigation

- Use the keyboard arrows $\boxed{\rightarrow}$, $\boxed{\downarrow}$, $\boxed{\leftarrow}$, and $\boxed{\uparrow}$ to navigate in the Finder.
- Use $\boxed{\text{click}}$ and $\boxed{\text{alt + click}}$ to navigate down and up, respectively, in the Network view.
- Use the search field in the Finder to directly navigate to nodes anywhere in the hierarchical structure.

Customization

- Click the button in the top right in the Network view window to toggle the Control panel.
- Filter the number of nodes by setting the link limit.
- Change the size and color scales in the corresponding tab.

Explanation

- The horizontal bars to the left of the node names in the Finder shows the flow volume of the nodes as a fraction of the whole network (color) and as a fraction of their parent nodes (size). The size is scaled logarithmically so that for each halving the flow is reduced by a factor ten. Thus, the two vertical lines within the bar measures, from left to right, 1% and 10% of the parent flow.
- The numbers after the modules in the Finder shows the maximum depth under that node in the tree. The same information is also encoded in the color of the arrow.
- The module names are automatically generated from their largest children, adding a new period for each level.
- Every module represents a cluster of nodes and the links between the modules represent the flow between the modules. The size of a module is proportional to the average time a random walker

spends on nodes in the module and the width of a link is proportional to the per step probability that a random walker moves between the modules.

3.2. The Infomap command line software

For large networks, the load and clustering time can be very long in the Flash-based Map Generator. To overcome this problem, the Infomap source code for clustering large networks can be executed from the command line. The code generates output files that can be loaded in the MapEquation web applications. Here we describe how to install and run Infomap from the command line with different options.

3.2.1. Installation

The latest Infomap code can be downloaded on www.mapequation.org/code.html. The source code is also available at Bitbucket[3]. Infomap is written in `C++` and can be compiled with `gcc` using the included `Makefile`. To extract the zipped archive and compile the source code in a Unix-like environment, open a terminal and type:

```
cd [path/to/Infomap]
unzip Infomap-0.13.5.zip
cd Infomap-0.13.5
make
```

Substitute **`[path/to/Infomap]`** with the folder where Infomap was downloaded, e.g. ~**`/Downloads`**. To be able to run on a Windows machine, you can install `MinGW/MSYS` to get a minimalist development environment where the above instructions will work. Please follow the instructions on MinGW - Getting Started[4] or download the complete MinGW-MSYS Bundle[5].

3.2.2. Running

To run Infomap on a network, use the following command:

```
./Infomap [options] network_data dest
```

The optional arguments can be put anywhere. Run **`./Infomap --help`** or see Sec. 3.2.3 for available options. The option **`network_data`** should point to a valid network file and **`dest`** to a directory where Infomap should write the output files. If no option is given, Infomap will assume an undirected network and try to partition it hierarchically.

3.2.3. Options

For a complete list of options, run **`./Infomap --help`**. Below we provide the most important options to Infomap:

```
------- Input -------
-i<o>, --input-format=<o> # Specify input format ('pajek' or 'link-list') to override format
    possibly implied by file extension.
-z, --zero-based-numbering # Assume node numbers start from zero in the input file instead of
    one.
-k, --include-self-links # Include links with the same source and target node. (Ignored by
    default.)
-c<p>, --cluster-data=<p> # Provide an initial two-level solution (.clu format).

------- Output -------
--tree # the hierarchy in .tree format. (default true)
--map # the top two-level modular network in the .map format.
--clu # the top cluster indices for each node.
--node-ranks # the calculated flow for each node to a file.
--btree # the tree in a streamable binary format.
--bftree # the tree including horizontal flow links in a streamable binary format.

------- Dynamics -------
-u, --undirected # Assume undirected links. (default)
-d, --directed # Assume directed links.
```

[3] https://bitbucket.org/mapequation/infomap
[4] http://www.mingw.org/wiki/Getting_Started
[5] http://sourceforge.net/projects/mingwbundle/files/latest/download

```
-t, --undirdir # Two-mode dynamics: Assume undirected links for calculating flow, but directed
     when minimizing codelength.
-y<f>, --self-link-teleportation-probability=<f> # The probability of teleporting to itself.
    Effectively increasing the code rate, generating more and smaller modules.

------- Miscellaneous -------
-2, --two-level # Optimize a two-level partition of the network.
-s<n>, --seed=<n> # A seed to the random number generator.
-N<n>, --num-trials=<n> # The number of outer-most loops to run before picking the best
    solution.
-v, --verbose # Verbose output on the console. Add additional 'v' flags to increase verbosity
    up to -vvv.
-h,--help # Prints help message with full list of commands.
```

Argument types are defined by:

```
<n> # a positive integer
<f> # a floating point number
<p> # a path to a file or directory
<o> # a string that matches one of the listed options
```

3.2.4. Examples

We begin with a simple example with the network file `ninetriangles.net` provided in the root directory of the source code. First we create a new directory where we want the put the results, and feed the destination to Infomap together with the input network and the option to **`-N 10`**, which tells the code to pick the best result from 10 attempts.

```
mkdir output
./Infomap ninetriangles.net output/ -N 10
```

Now Infomap will try to parse the file `ninetriangles.net` as an undirected network, try to partition it hierarchically, and write the best result out of 10 attempts to the output directory as a `.tree` file described in Sec. 3.3. In the second example specified below, Infomap will treat the network as directed and try to find the optimal two-level partition with respect to the flow. With this command, Infomap will also create an output `.map` file that can be visualized with the Map Generator, for example.

```
./Infomap ninetriangles.net output/ -N 10 --directed --two-level --map
```

As mentioned earlier, for viewing large networks in the Hierarchical Network Navigator, we recommend options **`--btree`** or **`--bftree`** to generate streamable binary `.btree` and `.bftree` files, respectively.

For acyclic or time-directed networks, such as article-level citation networks, we recommend the option **`--undirdir`**, which makes Infomap use a random walk model with movements both along and against link directions but encoding only along link directions. With the standard random walk model, a disproportional amount of flow will reach the oldest articles.

3.3. Network input and output formats

Here we specify all file formats that can be loaded as input or generated as output from the applications in the MapEquation software package.

3.3.1. Pajek's `.net` format

Network data with the format of Pajek's `.net` format can be loaded in all applications of the MapEquation software package. The Pajek format specifies both the nodes and the links in two different sections of the file. In the `.net` file, the network nodes have one unique identifier and a label. The definition of nodes starts with the line `*Vertices N`, where `N` is the number of nodes in the network. Labels or node names are quoted directly after the node identifier. The link section starts with the line `*Edges L` or `*Arcs L` (case insensitive), where `L` is the number of links. Weights can be given to nodes by adding a third column with positive numbers. Below we show an example network with six nodes and eight directed and weighted links.

```
# A network in Pajek's .net format
*Vertices 6
1 "Node 1" 1.0
2 "Node 2" 1.0
3 "Node 3" 1.0
4 "Node 4" 1.0
```

```
5 "Node 5" 1.0
6 "Node 6" 1.0
*Arcs 8
1 2 3.0
2 3 2.0
3 1 2.0
1 4 1.0
4 5 2.0
5 6 2.0
6 4 2.0
4 1 1.0
```

Pajek uses `*Edges` for undirected links and `*Arcs` for directed links. The MapEquation software package accepts both `*Edges` and `*Arcs` and the choice of load button in the user interface determines whether the algorithm treats the network as undirected or directed. Directed links have the form `from to weight`. That is, the first link in the list above goes from node 1 to node 2 and has weight 3.0. The link weight is optional and the default value is 1 (we aggregate the weights of links defined more than once). Node weights are optional and sets the relative proportion to which each node receives teleporting random walkers in the directed version of the code.

3.3.2. The link list format

The Hierarchical Network Navigator can, in addition to a Pajek `.net` file, also load a link list. A link list is a minimal format to describe a network by only specifying a set of links as shown below. Each line corresponds to the triad `source target weight`, which describes a weighted link between the nodes with specified numbers. The weight can be any non-negative value. If omitted, the default link weight is 1. The nodes are assumed to start from 1 and the total number of nodes will be determined by the maximum node number.

```
# A network in link list format
1 2 1
1 3 1
2 3 2
3 5 0.5
...
```

3.3.3. Pajek's `.clu` format

For a given network input file, it is also possible to specify the clustering of the nodes in all applications. The cluster information must be provided in Pajek's `.clu` format. In the web applications, the file can be loaded after the network by clicking $\boxed{\text{Load cluster data}}$. Infomap reads the cluster information with the option **`-c clusterfile.clu`** Pajek's `.clu` format is just a list of module assignments as shown below.

```
# A clustering in Pajek's .clu format
*Vertices 6
2
2
2
1
1
1
```

The cluster file above specifies that nodes 1-3 in the network belong to module 2 and that nodes 4-6 belong to module 1. Infomap generates a `.clu` file with the option **`--clu`**.

3.3.4. The `.map` and `.smap` formats

Information contained in the network and the cluster file together can be loaded in the web applications as a single `.map` file, which also include link and node information aggregated at the module level. The `.map` file begins with information about the number of nodes, modules, and links in the network, followed by the modules, nodes, and the links between modules in the network as shown below.

```
# A network clustering file in the .map format
# modules: 2
# modulelinks: 2
# nodes: 6
# links: 8
# codelength: 2.51912
*Directed
*Modules 2
1 "Node 1,..." 0.5 0.0697722
2 "Node 4,..." 0.5 0.0697722
*Nodes 6
1:1 "Node 1" 0.209317
1:2 "Node 3" 0.147071
1:3 "Node 2" 0.143613
2:1 "Node 4" 0.209317
2:2 "Node 6" 0.147071
2:3 "Node 5" 0.143613
*Links 2
1 2 0.0697722
2 1 0.0697722
```

This `.map` file also contains the codelength and the flow volumes of the nodes, and was generated with Infomap. In the output from Infomap, the names under `*Modules` are by default derived from the node with the highest flow volume within the module and `0.5 0.0697722` represent, respectively, the aggregated flow volume of all nodes within the module and the per step exit flow from the module. The nodes are listed with their module assignments together with their flow volumes. Finally, all links between the modules are listed in order from high flow to low flow. Infomap generates a `.map` file with the option **`--map`**.

The `.smap` file below corresponds to the `.map` file above with additional significance information.

```
# A network significance clustering file in the .smap format
# modules: 2
# modulelinks: 2
# nodes: 6
# links: 8
# codelength: 2.51912
*Directed
*Modules 2
1 "Node 1,..." 0.5 0.0697722
2 "Node 4,..." 0.5 0.0697722
*Insignificants 1
2 < 1
*Nodes 6
1:1 "Node 1" 0.209317
1:2 "Node 3" 0.147071
1;3 "Node 2" 0.143613
2:1 "Node 4" 0.209317
2:2 "Node 6" 0.147071
2;3 "Node 5" 0.143613
*Links 2
1 2 0.0697722
2 1 0.0697722
```

The `.smap` file contains the necessary information to generate a significance map in the Alluvial Generator. Compared to the `.map` file above, this file also contains information about which modules that are not significantly standalone and which modules they most often are clustered together with. The notation `2 < 1` under `*Insignificants 1` in the example above means that the significant nodes in module 2 more often than the confidence level are clustered together with the significant nodes in module 1. In the module assignments, we use colons to denote significantly clustered nodes and semicolons to denote insignificantly clustered nodes. For example, the colon in `1:1 "Node 1" 0.209317` means that the node belongs to the by flow largest set of nodes that are clustered together more often than the confidence level number of bootstrap networks. This `.smap` file was generated with code described in Sec. 3.1.2. For more information, see ref. (35).

3.3.5. The `.tree` format

The default output from Infomap is a `.tree` file that contains information about the identified hierarchical structure. The hierarchical structure in the .tree file below has three levels.

```
# A .tree file specifying a tree-level hierarchical structure
# Codelength = 3.48419 bits.
1:1:1 0.0384615 "7"
1:1:2 0.0384615 "8"
1:1:3 0.0384615 "9"
1:2:1 0.0384615 "4"
1:2:2 0.0384615 "5"
...
```

Each row begins with the multilevel module assignments of a node. The module assignments are colon separated from coarse to fine level, and all modules within each level are sorted by the total PageRank of the nodes they contain. Further, the integer after the last comma is the rank within the finest-level module, the decimal number is the steady state population of random walkers, and finally is the node name within quotation marks. Infomap generates a `.tree` file by default, or with the option **`--tree`**.

3.3.6. The `.btree` and `.bftree` formats

To be able to navigate the network as soon as the hierarchical structure has been loaded into the Hierarchical Network Navigator, we use a customized streamable format that includes the tree structure in a breath first order (`.btree` and `.bftree`), including the flow links of each sub-network (`.bftree` only). In this way, only a small part of the file has to be loaded to visualize the top structures, and the deeper structures can be loaded on demand. Infomap generates the `.btree` and `.bftree` files with the options **`--btree`** and **`--bftree`**, respectively.

4. Conclusions

This tutorial is meant to serve as a guideline for how to use the map equation framework for simplifying and highlighting important structures in networks. We have described several applications developed for analysis and visualization of large networks. However, we haven't covered all features and new software is under development. Ultimately, for maximal usability and user-friendly interface, we would like the web application to be as fast as the command line software. We are continuously investigating different solutions that become available with new web technology, and welcome all feedback. Therefore, we encourage you to contact us[6] if you have any questions or comments. Please visit www.mapequation.org for the latest releases and updates, including an up-to-date version of this tutorial.

References

1. Kanungo, T. *et al.* An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, 881–892 (2002).
2. Ward, J. H. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association* **58**, 236–244 (1963).
3. Blei, D. M., Ng, A. Y. & Jordan, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res.* **3**, 993–1022 (2003).
4. Dorogovtsev, S. N. & Mendes, J. F. F. *Evolution of Networks: From Biological Nets to the Internet and WWW* (Oxford University Press, 2003).
5. Barrat, A., Barthlemy, M. & Vespignani, A. *Dynamical Processes on Complex Networks* (Cambridge University Press, New York, NY, USA, 2008).
6. Newman, M. E. J. *Networks: An Introduction* (Oxford University Press, 2010).
7. Rosvall, M. & Bergstrom, C. Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. USA* **105**, 1118 (2008).
8. Lancichinetti, A. & Fortunato, S. Community detection algorithms: a comparative analysis. *Phys. Rev. E* **80**, 056117 (2009).
9. Aldecoa, R. & Marín, I. Exploring the limits of community detection strategies in complex networks. *Sci. Rep.* **3**, 2216 (2013).
10. Rosvall, M., Esquivel, A. V., Lancichinetti, A., West, J. D. & Lambiotte, R. Memory in network flows and its effects on community detection, ranking, and spreading. *Preprint at arXiv:1305.4807* (2013).
11. Fortunato, S. Community detection in graphs. *Physics Reports* **486**, 75–174 (2010).
12. Newman, M. E. & Girvan, M. Finding and evaluating community structure in networks. *Physical review E* **69**, 026113 (2004).
13. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008**, P10008 (2008).
14. Lancichinetti, A., Radicchi, F., Ramasco, J. & Fortunato, S. Finding statistically significant communities in networks. *PLoS ONE* **6**, e18961 (2011).
15. Karrer, B. & Newman, M. E. Stochastic blockmodels and community structure in networks. *Physical Review E* **83**, 016107 (2011).
16. Gopalan, P. K. & Blei, D. M. Efficient discovery of overlapping communities in massive networks. *Proceedings of the National Academy of Sciences* **110**, 14534–14539 (2013). PMID: 23950224.
17. Peixoto, T. P. Hierarchical block structures and high-resolution model selection in large networks. *Preprint at arXiv:1310.4377* (2013).

[6] www.mapequation.org/about

18. van Dongen, S. *Graph Clustering by Flow Simulation.* Ph.D. thesis, University of Utrecht (2000).
19. Lancichinetti, A., Fortunato, S. & Radicchi, F. Benchmark graphs for testing community detection algorithms. *Physical Review E* **78**, 046110 (2008).
20. Fortunato, S. & Barthelemy, M. Resolution limit in community detection. *Proceedings of the National Academy of Sciences* **104**, 36–41 (2007).
21. Waltman, L. & Eck, N. J. A new methodology for constructing a publication-level classification system of science. *Journal of the American Society for Information Science and Technology* **63**, 2378–2392 (2012).
22. Traag, V. A., Van Dooren, P. & Nesterov, Y. Narrow scope for resolution-limit-free community detection. *Physical Review E* **84**, 016114 (2011).
23. Kawamoto, T. & Rosvall, M. The map equation and the resolution limit in community detection. *Preprint at arXiv:1402.4385* (2014).
24. Lambiotte, R. & Rosvall, M. Ranking and clustering of nodes in networks with smart teleportation. *Phys. Rev. E* **85**, 056107 (2012).
25. Golub, G. H. & Van Loan, C. F. *Matrix computations*, vol. 3 (JHU Press, 2012).
26. Schaub, M. T., Lambiotte, R. & Barahona, M. Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation. *Phys. Rev. E* **86**, 026112 (2012).
27. Huffman, D. A. *et al.* A method for the construction of minimum redundancy codes. *Proceedings of the IRE* **40**, 1098–1101 (1952).
28. Shannon, C. E. & Weaver, W. A mathematical theory of communication (1948).
29. Brin, S. & Page, L. The anatomy of a large-scale hypertextual web search engine. *Comput. Networks ISDN* **30**, 107–117 (1998).
30. Rosvall, M. & Bergstrom, C. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLoS ONE* **6**, e18209 (2011).
31. Esquivel, A. & Rosvall, M. Compression of flow can reveal overlapping-module organization in networks. *Phys. Rev. X* **1**, 021025 (2011).
32. Newman, M. E. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review E* **64**, 016132 (2001).
33. Newman, M. E. The structure and function of complex networks. *SIAM review* **45**, 167–256 (2003).
34. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M. & Hwang, D.-U. Complex networks: Structure and dynamics. *Physics reports* **424**, 175–308 (2006).
35. Rosvall, M. & Bergstrom, C. Mapping change in large networks. *PLoS ONE* **5**, e8694 (2010).