MULTISCALE COMMUNITY DETECTION USING MARKOV DYNAMICS

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAI‘I AT MĀNOA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

AUGUST 2014

By
Ali Altunkaya

Thesis Committee:

Daniel D. Suthers, Chairperson
J.B. Nation
David N. Chin

We certify that we have read this thesis and that, in our opinion, it is satisfactory in scope and quality as a thesis for the degree of Master of Science in Computer Science.

THESIS COMMITTEE

_____
Chairperson

_____

_____

Acknowledgments

I would like to acknowledge the constant help, advice, and support of Daniel D. Suthers, my principal advisor and committee chair for this thesis. I would like to thank him for the guidance and mentorship in all aspects of my thesis, which I learned about the research process and the technical writing. Especially I am grateful for his encouragement, patience and the friendly lab environment during the thesis.

I would like to thank my co-advisor J.B. Nation for his support, guidance and encouragement from the beginning of this project. I was fortunate to attend his lectures and discussions in his office, which helped me to understand mathematical concepts in my thesis.

I would also like to thank my committee member David N. Chin for his continuous and friendly support during the thesis. I am thankful for his positive point of view and good will.

Finally, I am very grateful for my family and my parents Fadime and Aliihsan. Without their support and encouragement I would not have completed this project.

Abstract

Complex networks is an interdisciplinary research area getting attention from a variety of disciplines including sociology, biology, and computer science. It studies the properties of complex systems that may have many functional or structural subunits. Community detection algorithms are one of the major approaches to analyse complex networks by finding these intermediate-level subunits called modules or communities. Furthermore, some networks may have multilevel or overlapping community structures.

InfoMap is one of the best performing algorithms that finds the communities of a network by compressing the flow of information [25, 17]. In this work, we introduced Markov Dynamics to the InfoMap in order to detect communities at multiscales. Although Markov Dynamics have been applied to the InfoMap for undirected networks before [28], this was the first application of Markov Dynamics to the InfoMap for directed networks. Additionally, we added a feature to detect overlapping nodes using the compression of flow on the boundary nodes, similar to the approach described in [9] before.

These two features were combined into the InfoMap for directed networks. We evaluated these two features on synthetic networks and benchmark graphs. We have comparable results with the Hierarchical InfoMap [26] for multilevel community detection, but improvement in runtime is needed. We evaluated the overlapping feature by its own ability, and found that it can detect overlapping nodes for networks with sparse overlaps.

Contents

List of Figures

Chapter 1

Introduction

The whole is more or less than the sum of its parts. The nature of interactions between the parts affects the functions and behaviours of the whole. Many objects from humans to genes to computers interact with each other in nonrandom ways that generate social, biological or technological networks [24]. For quantitative analysis, these networks can be represented using graphs where each element is a node and the interactions between nodes are links. To understand the functions and behaviours of these complex networks, we need to find intermediate-level modules.

Although each individual member of these networks has a particular function or behaviour, some complex functions or behaviours can only be accomplished by a group of members. For example, a group of chemicals can form a metabolic pathway, a group of people can form a company or clique, a group of computers can form a subnet. Furthermore groups interact with each other to become a whole. These groups are also known as *modules* or *communities* in complex networks lingo. Although there is no consensus on the definition of a community [1], a *community* can be viewed as a group of nodes that has a shared identity or that are more related to each other than others. Modules or communities are intermediate-level structures between the microscopic level of nodes and the macroscopic panorama of the whole network [24].

In real life, sometimes we have the full or partial view of a complex network, but we do not know the modules or communities. For example, we may have the full graph of a social network website that shows the members and the interactions between them, but we don't know the communities. The goal of a community-detection algorithm is to find the modules or communities of a network using only the interactions between its members, which may be a (un)weighted and (un)directed graph.

We would like to note a small difference between two well-known problems, graph partitioning, and community detection. Both problems are similar, but in graph partitioning we need

to partition the graph into a number of modules that is given in advance. In community detection, we don't know the number of modules in advance. We need to find the number of modules and the partition of the network into modules at the same time [21]. Of course, this leads to different applications. Graph partitioning is used to match a graph to external criteria such as a set of processors. Community detection is used to find the "natural" structure of a graph.

Is there only one medium-level between the microscopic-level of nodes and the whole network? Often, complex architectures have multi-level structure. The modules at different scales may be hierarchical: nodes become groups of nodes, groups join and become supergroups, and so-forth [30]. But it is not necessary to be hierarchical. Sometimes there may be different community organizations at different scales, such as modules of the same network from different perspectives or aspects. It is like looking at a 3d object from different aspects. Sometimes it is useful to find communities at multiscales.

Chapter 2

Background

Giving a complete literature review about complex networks, its metrics, measures and community detection algorithms is out of the scope of this study. We would like to refer the interested reader to the more comprehensive and popular reviews and books [10, 21]. In this section, we are going to explain some important milestones of the community-detection algorithms from our subjective point of view and also elaborate how these selected algorithms constitute the base of our research plan.

2.1 Hierarchical Clustering

One of the traditional methods for community detection is hierarchical clustering. This method is an agglomerative approach. At the beginning, it assumes each node as a separate cluster, and computes similarities between all pairs of nodes within the network. Then it forms the communities by merging two nodes starting from highest to the lowest score [12]. After each merge process, it recomputes the similarity between the newly formed group and the rest of the nodes or clusters. When we choose the similarity measure, finding similarity between two nodes is straightforward. But how can we find similarity between a node and a cluster or between two clusters? There are three approaches for this purpose: the single-linkage approach uses the highest similarity score of two nodes between two groups, the complete-linkage approach uses the lowest similarity score of two nodes, and the average-linkage approach takes the average of all similarities between groups [21]. Eventually, each of these approaches will give us a dendrogram that shows the nested set of communities.

Similarity measures between two nodes are only determined by the network structure. There are many metrics to measure similarity between two nodes. For example cosine similarity is

one of those structural similarities. It uses the number of common neighbors between two nodes i and j. But only counting the number of common neighbors is not a good measure: we need to normalize it too. Cosine similarity normalizes the number of common neighbors, n_{ij}, with the geometric mean of their degrees k_i and k_j, $\theta_{ij} = q_{ij}/\sqrt{k_i * k_j}$. Cosine similarity produces values between 0 and 1. If both nodes have the same neighbors $\theta_{ij} = 1$, and if they have no common neighbors $\theta_{ij} = 0$ [21].

The choice of algorithm (single-linkage, average-linkage or complete-linkage) and the choice of similarity measure dramatically affects the quality of the clustering. Unfortunately there is no best choice. In most cases, they are chosen with experimental studies with the network at hand [21].

Hierarchical clustering finds the edges that are most central to the communities. It detects communities starting from the core to the periphery. Similarity metrics give higher scores to the edges more central to the communities than the edges less central. Because of that, the nodes that are connected with a single link to the rest of the network are added to the dendrogram later and become single-node clusters. This is one of the disadvantages of the hierarchical clustering: it results in many singleton communities [12].

2.2 Divisive Algorithms

Instead of focusing on the edges that are at the core of the communities, divisive algorithms try to find the edges that are on the boundaries of the communities. Girvan&Newman proposed the popular edge-betweenness metric to find the edges on the boundaries [12]. First, they compute the edge-betweenness values for each edge. For each edge, they count the number of shortest-paths that goes through the edge. If an edge is on the boundary between communities, then it is like an intersection point on a road and many shortest path routes run along it, thus it will have a high edge-betweenness score. Otherwise, if an edge is at the core of a community its edge-betweenness value will be low, since shortest paths between pairs of vertices are likely to use other edges.

Once we have the edge-betweennes values, the algorithm runs as a top-down approach. At the beginning, the whole network is one community. Then it deletes the edges one by one from the highest edge-betweennes to the lowest one. Eventually it will give us a dendrogram, similar to the hierarchical clustering.

The biggest problem of this method is that, it is NP-complete and not feasible even for small networks with hundreds of nodes. Even worse is that, to our knowledge no effective greedy algorithm has been developed for this method, which makes this algorithm impractical. Even if one had been developed, it would not be useful unless it were provably an approximation to optimal within a certain bound.

2.3 Modularity

Both hierarchical clustering and divisive algorithms output a dendrogram representing nested sets of components. Each level of the dendrogram results in a different community organization. A natural question is where should we cut the dendrogram? Which one of these possible community partitions is the best? At first, Newman and Girvan proposed the modularity metric as a measure for strength of the community structure for slices at different places [22]. Later, Newman used modularity as a standalone community-detection algorithm, a different approach from the edge-betweenness algorithm [20]. The reason is that, although the accuracy of modularity is not as good as the edge-betweenness and modularity optimization is NP-complete like the edge-betweenness algorithm, it is possible to develop greedy algorithms for modularity optimization that have good empirical approximations [6, 2]. Thus, it allows us to analyze very large networks with millions of nodes.

Modularity assumes that the best partition of the network is the one that has more edges within the communities and fewer edges between the communities compared to the random network model with the same size and degree [24]. Although modularity can be applied to weighted and directed networks, we will consider the unweighted and undirected network depicted in Fig. 2.1A. It is partitioned into three communities and the transition matrix for the communities is given in Fig. 2.1B. This matrix describes how the modules are connected by showing the number of end points of the edges that make the indexed transition either within or between communities. If we randomly distribute the edges to the same community division, then the transition matrix of the communities would be like in Fig. 2.1C. The modules have the same size and degree but random connectivity pattern. Modularity is the fraction of edges within the communities for the actual network minus the one for the random network.

In Fig. 2.1B-C the diagonal vector, in, gives the indegree (the number of degrees within the community) of that community, the external vector, tot, gives the total degree of that community, and $2m$ is the total degree of the network. For each community i, in_i/tot_i gives us the ratio of

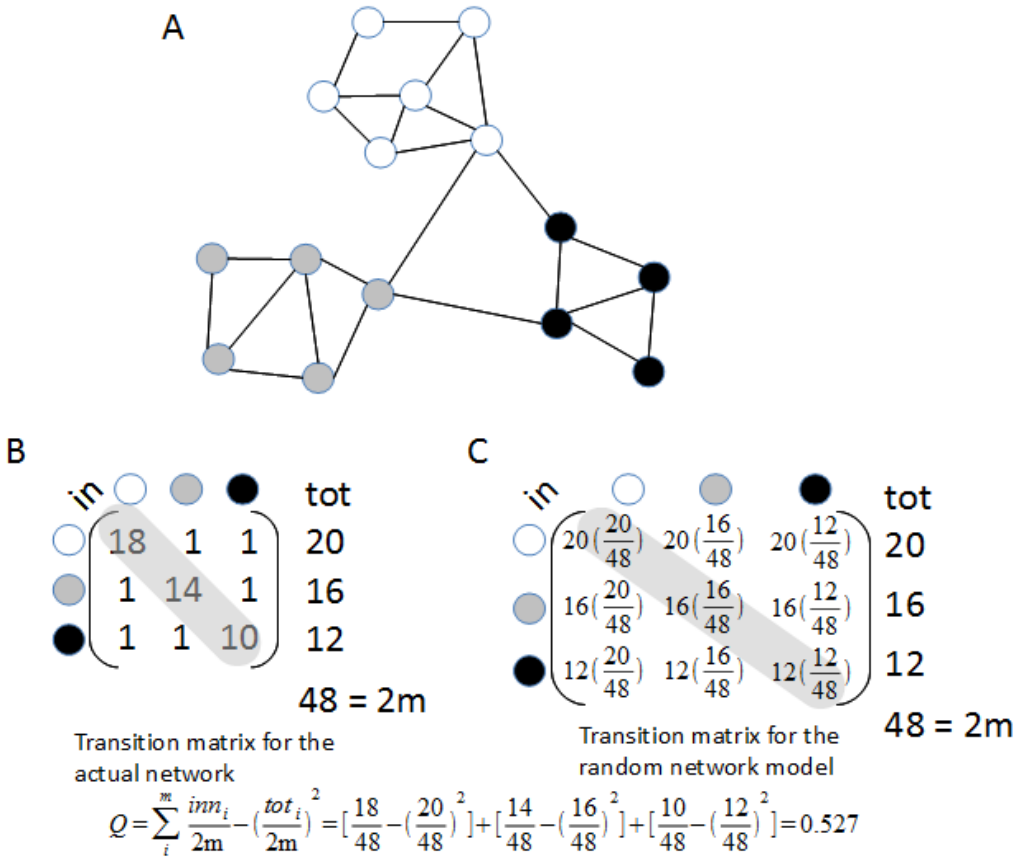


Figure 2.1: (A) A sample network partitioned into three communities. (B) The transition matrix of the communities for the actual network. (C) The transition matrix of the communities for the random network model.

degrees within the community to the total degree of that community, and $tot_i/2m$ gives us the relative size of that community to the whole network. Therefore $(in_i/tot_i) * (tot_i/2m) = in_i/2m$ gives us the normalized value for the fraction of edges within the community i [14].

Let's calculate the fraction of edges for both the random network model and the actual one. For the random network model, in_i/tot_i equals $tot_i/2m$ for obvious reasons of homogenity, thus we can write the normalized value for the fraction of edges within the community as $(tot_i/2m)^2$. In the case of the actual network, $(in_i/tot_i) * (tot_i/2m)$, we can cancel out the tot_i, so $in_i/2m$ will give us the fraction. We can formulate the modularity, Q, as:

Q = (fraction of edges within the communities) - (expected fraction of such edges) [14]

$$Q = \sum_{i=1}^{m} \frac{in_i}{tot_i}\frac{tot_i}{2m} - \frac{tot_i}{2m}\frac{tot_i}{2m} = \sum_{i=1}^{m} \frac{in_i}{2m} - (\frac{tot_i}{2m})^2 \tag{2.3.1}$$

One shortcoming of modularity is the resolution limit [11]. Modularity works well if the modules are similar in size and degree. Because of the $(2m)^2$ denominator, the choice of partition depends on the total number of links in the network, and it cannot detect communities smaller than a scale that depends on the total size of the network [24].

2.4 Information-theoretic methods

Rosvall & Bergstrom introduced two information-theoretic approaches, infoMod and InfoMap, for community detection [24, 25]. In this section, we will briefly explain these two methods.

2.4.1 InfoMod

InfoMod identifies the modules of an undirected network by finding the optimal compression of its topology [24]. Infomod describes a network that has n nodes partitioned into m modules, using an assignment vector a that shows the assignment of nodes to modules, and a module matrix M that shows the connections between the modules as shown below:

$$Y = \left\{ a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}, M = \begin{pmatrix} l_{11} & \cdots & l_{1m} \\ \vdots & \ddots & \vdots \\ l_{m1} & \cdots & l_{mm} \end{pmatrix} \right\} \tag{2.4.1}$$

Assume we compress the actual network in Fig. 2.2A into three modules like Fig. 2.2B. From an information-theoretic perspective, what is the information (uncertainty) necessary to recover the original network from the compressed one? InfoMod finds $H(Z)$, the information (uncertainty) of the compressed representation by calculating the number of possible network estimates from it.

$$H(Z) = \log \left[\prod_{i=1}^{m} \binom{n_i(n_i-1)/2}{l_{ii}} \prod_{i>j} \binom{n_i n_j}{l_{ij}} \right] \tag{2.4.2}$$

The first term gives the number of different ways to connect nodes within each module i, where each module has n_i nodes and l_{ii} links. The second term gives the number of different ways to connect m modules with each other, where l_{ij} is the number of links between modules i and j.

If we know the number of communities in advance, we can find the optimal partition by minimizing H(Z). But, usually we need to find the number of modules and the assignment of the nodes to modules at the same time. To accomplish this, infoMod uses the MDL (Minimum

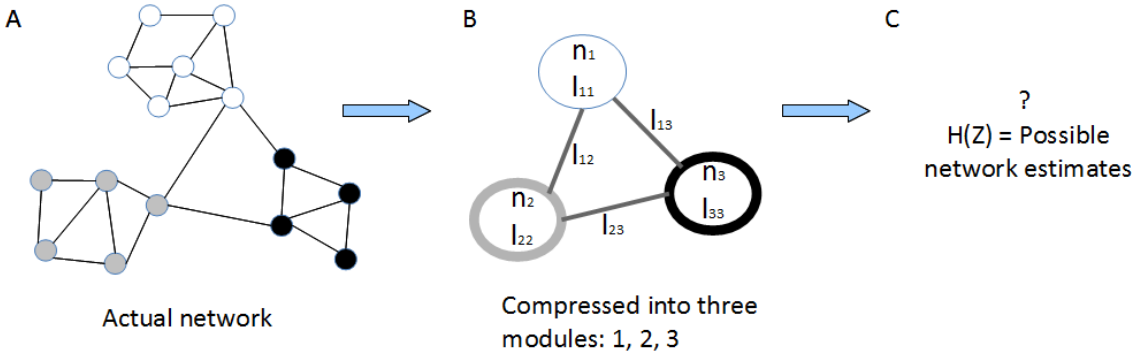


Figure 2.2: Information-theoretic modularity. Drawing is inspired by the figure from [24]. (A) The original network. (B) Compressed into three modules with n_i nodes. There are l_{ii} links inside each community, and l_{ij} links connect community i and j. (C) The number of possible networks that is estimated using this compressed description.

Description Length) principle. The MDL principle helps us to choose a model without overfitting the particular dataset [24]. It balances the information (uncertainty) in the description H(Z), with the length of the description in bits L(Y). InfoMod finds the length of the description in bits as shown below:

$$L(Y) = \log \left[m^n * l^{\frac{m(m+1)}{2}} \right] = n \log m + \frac{1}{2} m(m+1) \log l \tag{2.4.3}$$

The first term gives the size to encode the assignment vector in bits and the second term gives the size to encode the module matrix in bits. Eventually, infoMod seeks to minimize the sum of uncertainty within the description and length of the description in bits.

$$H(Z) + L(Y) \tag{2.4.4}$$

The sum balances the two, since increasing one decreases the other and vice versa.

2.4.2 InfoMap

Both modularity and infoMod make use of the topology of the network while finding modules, but applying different approaches. These modularity based techniques assume that a module should have more connections within the module and fewer connections between them. They work well for clique-like modules in undirected networks, but may not work in directed networks if there is a flow. Before explaining the details of the InfoMap, we will explain the main paradigm-shift between modularity based methods and flow based methods such as InfoMap.

Both modularity and infoMod uses the pairwise interactions between the nodes, thus intuitively finding the modules that have more pairwise interactions (connections or edges) within the module and fewer between the modules [23]. Unlike these methods, InfoMap uses the flow within the network, thus intuitively finding the modules that have more flow inside the community. In other words, if an entity enters a community, it will spend more time within that community before exiting.

Let us explain this with an example. If a network is generated by the pairwise interactions between nodes, then there is no direction and often no weight. There is no interdependency: the connection between nodes A and B does not have an effect on the connection between B and C. Also there is no flow: each node is a sink or source [23]. For example, we can generate an Airline Transportation Network, by placing an edge between airports if there is a route between them. This will result an undirected and unweighted network, where there is no interdependency. Modularity based methods probably works well for these kind of networks. On the other hand, we can also generate another Airline Transportation Network which shows the passenger flow between airports. In this case, we put directed edges between nodes (airports) that are weighted according to the number of the passengers. Now the nodes are interdependent on each other, because the number of incoming passengers affect the number of outgoing passengers. Because of the conservation of flow, there is no sink or source node, information just flows within this directed-weighted network. InfoMap works well for these kind of network.

We can give another example in social networks to clarify the distinction between modularity based methods and the flow based method, InfoMap. If we generate a friendship network by putting an edge between two people who are friends (a pairwise interaction), it will result an undirected, and probably unweighted network. There is no interdependency between the nodes, since a friendship between A-B cannot effect the friendship of B-C in a quantitative way. This network can be analyzed using modularity based methods. Let's try to generate a gossip network between people. Assume we know the number of gossips each person tells to another person, and we generate a directed and weighted network between people using this information. Now, the information (gossip) flows between people and the nodes are interdependent: if A tells 10 gossips to B, then probably B will tell some of them to his other friends C or D, and a gossip usually stays and flows within the same module and rarely exits from this module and enters to another one. InfoMap works better for this kind of network.

Now, we will explain the technical details of InfoMap. InfoMap finds the modules of a directed and weighted network by compressing the information flow on the network. InfoMap uses

the probability flow of random walks on network as a proxy for information flows [25]. If a random walker surfs on the network, it produces a trajectory as shown in Fig. 2.3. InfoMap uses two kinds of regularities to compress the trajectory of a random walker on the graph: using short codewords for frequently visited nodes, and using two-level codewords, so we can reuse the same codewords within different modules [9].

Fig. 2.3 illustrates how InfoMap works on a 15-node network. Fig. 2.3A shows the trajectory of a random walker for a particular 18-step walk. The basic approach gives unique names to each node, and the average length per step is $log_2 15 = 3.9$ bits. In Fig. 2.3B, we apply Huffman coding to compress the same trajectory by using short codewords for frequently visited nodes, and long codewords for rarely visited ones. It reduces the average length per step to 3.88 bits. Finally, in Fig. 2.3C, two-level codewords are applied to reuse the same codewords within different modules. The intuition comes from a map metaphor: the same street names (Main, Broadway etc) can be reused in different cities (New York, Los Angeles, Honolulu). This encoding reduces the average length per step to 3.27 bits.

We just showed this Huffman coding example to illustrate how InfoMap works on a concrete example. Actually, InfoMap does not devise actual codewords for nodes. It only computes the theoretical lower limit for the compression of the trajectory of a random walker on the network [23]. Before the details, we will give the definition of stationary distribution: If we assume that the time spent on each node is equal at initial distribution, and the random walker traverses the network infinitely according to weights, then the time spent on each node will become stable and it is called stationary distribution. The stochastic process reaches the stationarity, the condition of being stationary, where the statistical properties of the stochastic process become invariant with respect to time.

InfoMap computes the theoretical lower limit only by using the ergodic node visit frequencies: If the random walker surfs on the network infinitely, then the time spent on each node i will become stationary (stable), and its proportion to the whole network is called the ergodic node visit frequency of node i, p_i. How can we make sure that a directed network has a unique stationary distribution? InfoMap introduces a small teleportation probability λ that links each node to every other node with a positive probability, so the random walker does not get stuck in a module, and that makes the Markov chain irreducible [1] and aperiodic. According to the Perron-Frobineous theorem, an irreducible and aperiodic Markov chain has a unique stationary distribution [23]. By the way,

[1] A Markov chain is irreducible if there is a non-zero probability of transitioning (even if in more than one step) from any state to any other state.

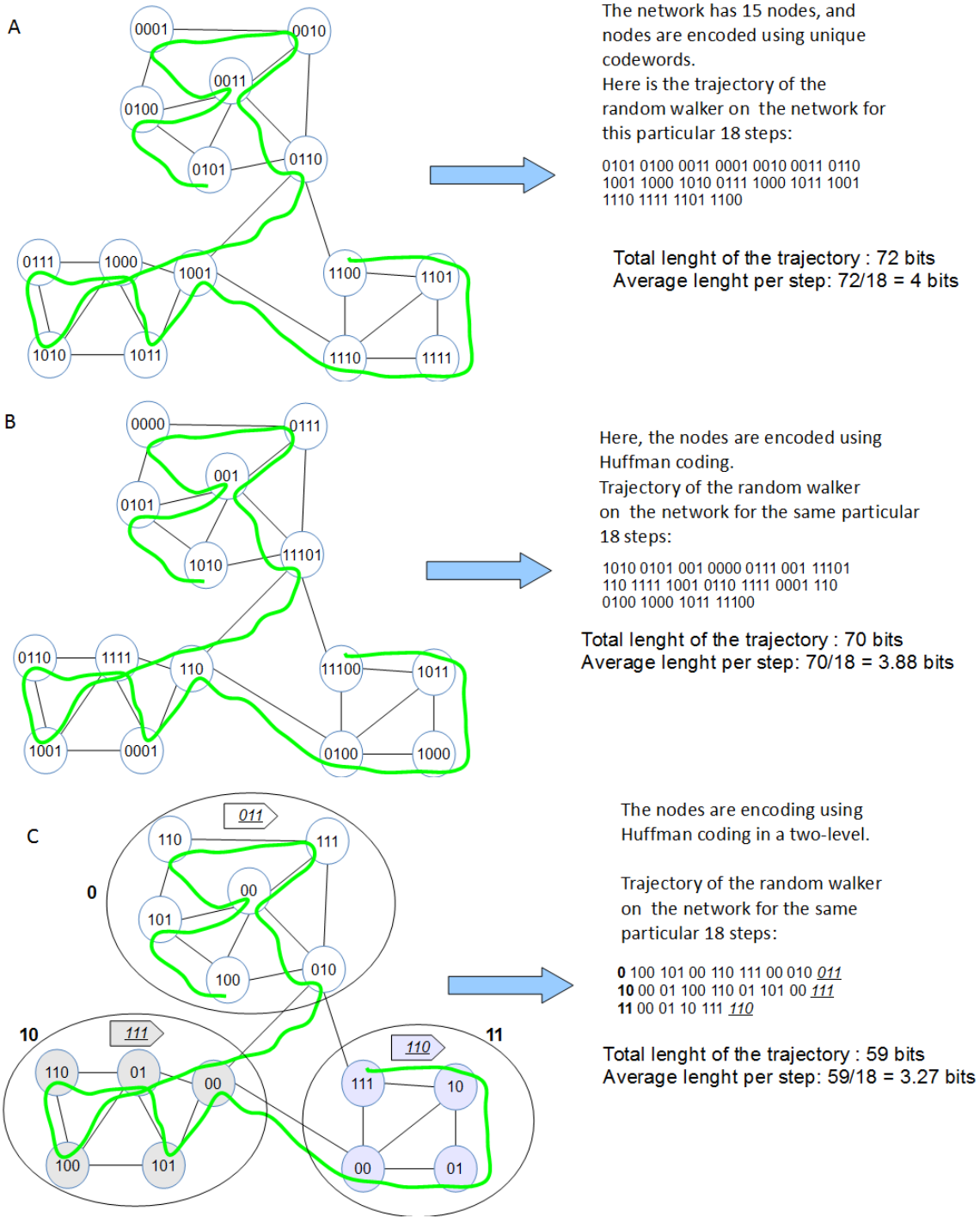


Figure 2.3: Illustration of the InfoMap. Drawing is inspired by the figure from [25]. (A) The basic approach for encoding nodes. (B) Encoding nodes using Huffman coding. (C) Encoding nodes using Map metaphor + Huffman coding.

for connected and undirected networks, the corresponding Markov processes by construction have a unique stationary distribution. InfoMap applies the map metaphor and reuses the same codewords within different communities. To accomplish the map metaphor, InfoMap adds an extra exit codeword q_α for each community, to indicate that the random walker is leaving the community α and enters another community. q_α is the probability of the random walker leaving community α, and can be computed as:

$$q_\alpha = \sum_{i \in \alpha} \sum_{j \notin \alpha} p_i M_{ij} \tag{2.4.5}$$

We will also need the sum of all exit probabilities q, and the sum of ergodic node visit frequencies p^α for each community α. We need to weight entropies of these codebooks with their relative usage.

$$q = \sum_{\alpha=1}^{m} q_\alpha \tag{2.4.6}$$

$$p^\alpha = q_\alpha + \sum_{i \in \alpha} p_i \tag{2.4.7}$$

InfoMap assumes that there is one index codebook to encode movements between communities and m module codebooks to encode movements within each community. Now the entropy $H(P^\alpha)$ of movement within each community α can be computed using ergodic node visit frequencies within that community and the extra exit codeword:

$$H(P^\alpha) = -\frac{q_\alpha}{p^\alpha} log(\frac{q_\alpha}{p^\alpha}) - \sum_{i \in \alpha} \frac{p_i}{p^\alpha} log(\frac{p_i}{p^\alpha}) \tag{2.4.8}$$

In an ergodic flow, the incoming flow equals the outgoing flow, therefore the exit probability q_α of a module can also be considered as its ergodic module visit frequency. Thus, the entropy of movement between communities can be computed using the exit probabilities of modules, as in formula (2.4.9):

$$H(Q) = -\sum_{\alpha=1}^{m} \frac{q_\alpha}{q} log(\frac{q_\alpha}{q}) \tag{2.4.9}$$

Finally InfoMap balances the entropy of movement between the modules, $H(Q)$, with the entropy of movement within the modules, $H(P_\alpha)$. Entropies are also weighted with the rate that each codebook is used: p^α and q. The best partition of the network is the one that minimizes the InfoMap equation (2.4.10) best.

$$L(M) = qH(Q) + \sum_{\alpha=1}^{m} p^\alpha H(P^\alpha) \tag{2.4.10}$$

The original InfoMap equation has two constraints: the code structure should have two levels and a node can only be assigned to one module. Subsequent modifications to InfoMap by Rosvall et.al. finds hierarchical multilevel communities by releasing the first constraint and finds overlapping communities by releasing the second constraint [26, 9]. Also, InfoMap has been modified to find overlapping communities by clustering links instead of nodes [13], using Ahn et al.'s approach to overlapping community detection [1].

Unlike modularity, Infomap is immune to the resolution limit: there is no lower limit at which InfoMap cannot detect communities. But, InfoMap has a field of view limit, an upper scale that does not allow it to detect communities with large effective diameters. In these cases, InfoMap overpartitions the network [27, 28].

2.5 Markov Dynamics

Barahona et al. made important observations after studying modularity and InfoMap [8, 27, 28]:

- Both Infomap and Modularity are one-step methods, and cannot detect communities with large effective diameter.
- They neglect the internal structure of the modules, and the actual connectivity patterns between modules.
- They are biased to find clique-like modules. Not all real-world communities are clique like: some of them are ring-like or star-like communities.

To overcome these issues and to reveal community structure at multiple scales, Barahona et al. introduced Markov dynamics to both modularity and InfoMap. This method is called Stability. Stability uses a time-dependent transition matrix of the Markov process, and finds communities at multiple scales by scanning through all times. Since Markov dynamics is a diffusion process, it slowly and implicitly makes use of the full-structure of network and the actual connectivity patterns [27, 28]. For intuition, see Fig. 2.4 for a discrete-time Markov dynamics. As we can see in Fig. 2.4, the visibility from node 7 is marked by an outline and it increases with "time".

Some complex networks have multiscale structure. A multiscale structure can be hierarchical such that a network is composed of communities, and communities have their own subcommunities. Or a network may have non-hierarchical multiscale architecture, in which different

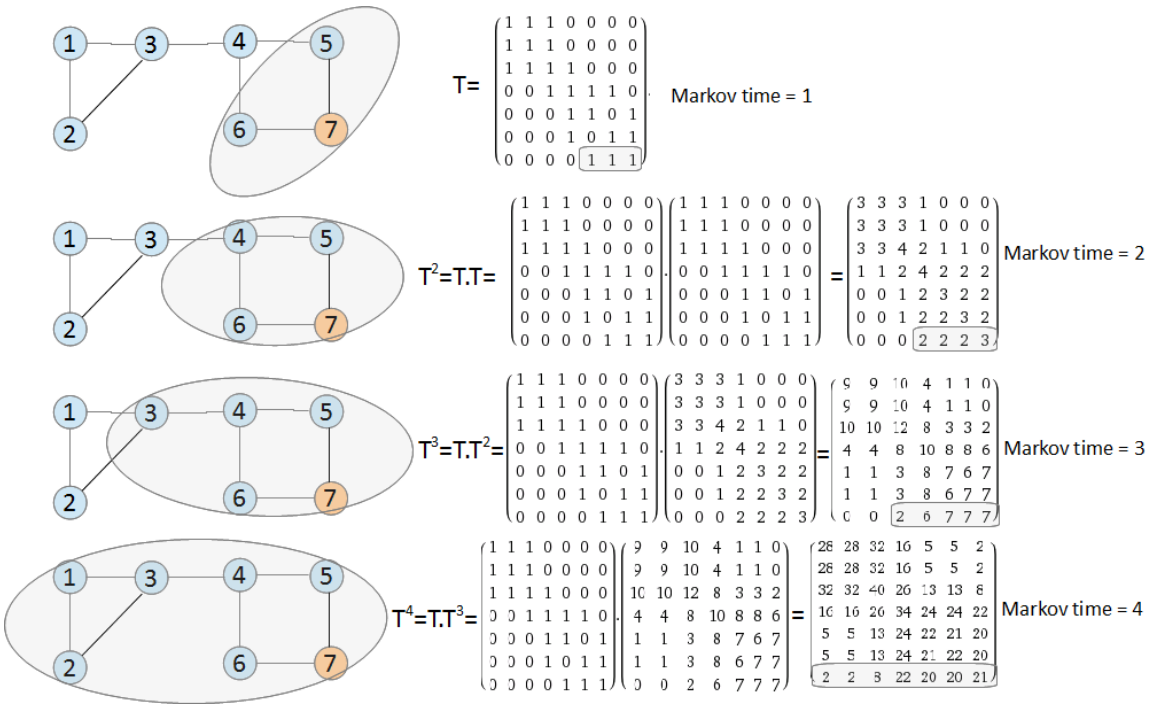


Figure 2.4: Markov dynamics is a diffusion process. Assume we have an undirected and unweighted graph and a matrix that shows field of view at different Markov-times. If you live at node 7, your field of view increases as Markov time increases.

communities can be found by looking at the same network from different perspectives. Stability has the potential to find communities at multiscales for both hierarchical and non-hierarchical community structures.

The purpose of stability is not only to find communities at multiple scales. Sometimes, the network may have only one community structure but it cannot be detected using one-step methods because of its large diameter (see Fig. 2.6 for ring communities). In this case, Stability uses Markov dynamics as a zooming lens to detect community structure. Both InfoMap and modularity can be considered as particular versions of Stability at Markov time 1 [27, 28]. The community structure cannot be detected at Markov time 1, but can be detected at larger Markov times.

Now, we will show the main idea of how they introduced Markov dynamics to modularity and InfoMap equations. For clarity issues, we will show Markov dynamics for discrete-time, but it can be easily adjusted to continuous-time Markov dynamics.

2.5.1 Markov Dynamics introduced to Modularity: Stability

Modularity is based on the structural properties of network, whereas Stability is based on the dynamics that occur on the network and uses the intimate relation between structure and dynamics to find communities at multiscales [8]. To compute modularity, we only need the adjacency matrix, but since Stability is based on flows of probability, we need the transition matrix of the graph to compute Stability. At Markov time equals 1, both Stability and Modularity are equal to each other. The modularity function finds the difference between the fraction of edges within the communities and the expected fraction of such edges in a random network [14]:

Q = (fraction of edges within the communities) - (expected fraction of such edges)

$$Q = \sum_{i=1}^{m} \frac{in_i}{tot_i} \frac{tot_i}{2m} - \frac{tot_i}{2m} \frac{tot_i}{2m} = \sum_{i=1}^{m} \frac{in_i}{2m} \frac{tot_i}{2m} - (\frac{tot_i}{2m})^2 \tag{2.5.1}$$

The final form of the modularity equation canceled out the tot_is within the first term. If we don't cancel out that term, here is the modularity:

$$Q = \sum_{i=1}^{m} \frac{in_i}{tot_i} \frac{tot_i}{2m} - \frac{tot_i}{2m} \frac{tot_i}{2m} \tag{2.5.2}$$

We can rewrite the Modularity formula using matrices, instead of summation. Summation goes from 1 to m, for m clusters. Instead we can use H, an Nxm 0-1 matrix, that shows which node belongs to which community. Additionally, we can replace the $tot_i/2m$ with vector π, the stationary distribution of a random walker, because for undirected graphs the stationary distribution

of a random walker at a node can be estimated by dividing the degree of that node by the total degree of the network. Π is the diagonal matrix of the vector π. Since we are using matrices, the square of $tot_i/2m$ can be obtained by multiplying the π by its transpose.

$$Q = \sum_{i=1}^{m} \frac{in_i}{tot_i} \frac{tot_i}{2m} - \frac{tot_i}{2m} \frac{tot_i}{2m} \tag{2.5.3}$$

$$R_t = H^T(\Pi M^t - \pi^T \pi)H \tag{2.5.4}$$

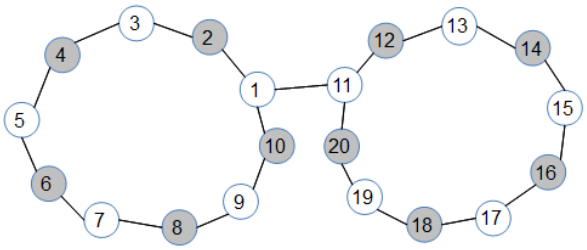
R_t = (the probability that a random walker starts and ends in the same community after t steps) - (the probability that two independent random walkers start and end in the same community at stationarity)

Scanning through time is an essential part of Stability, and is called Markov sweeping. If we want to find the stability of a particular partition H at time t, we need to compute Stability for all times up to t, and choose the minimum value of the trace up to time t [8]. We are taking the trace, a sum of diagonals, because Stability seeks the probability that a random walker starts and ends in the same community, which corresponds to diagonal elements of the matrix. Markov Sweeping makes the equation stable, because if the stability value of a particular partition fluctuates at different Markov times up to t, it chooses the minimum among them. Why? Because in a good partition, the random walker should remain within the same community for all times up to t. Stability is large only if its values are large for all times up to t. For example in Fig. 2.5, the R(t) is large for times 2, 4, and 6 but low for times 1, 3, and 5, because usually there are two steps between the members of a community. For example a random walker in node 3, stays in the same community after 2, 4, or 6 steps, but enters into another community after 1, 3, or 5 steps. In this case we assign the min value of R(t) for times up to t. This approach allows us to assign low stability to partitions that fluctuate over times up to t. For example, if there is a high probability of leaving the community at time 1, but also there is a high probability of coming back to it later at time 2. For discrete-time, we are doing Markov sweeping explicitly by taking the minimum of values for all times up to time t. For continuous-time, we don't need to do Markov sweeping explicitly by taking the minimum of values for all times up to t, because a continuous-time Markov process decreases monotonically.

$$r(t; H) = \min_{0<s<t} trace[R_s] \tag{2.5.5}$$

Of course, for each Markov time t, Stability tries to find the partition that maximizes the objective function.

$$r(t) = \max_{H} r(t; H) \tag{2.5.6}$$



t=Markov Time	Discrete time Rt	min(Rt)
1	-0.4535	-0.4535
2	0.4354	-0.4535
3	-0.4112	-0.4535
4	0.3966	-0.4535
5	-0.3783	-0.4535
6	0.3655	-0.4535

Figure 2.5: This network partitioned into two communities, white and gray communities. It is not a good partition but it is a good case to show why we need Markov Sweeping.

Both modularity and InfoMap cannot find the true solution for ring communities shown in Fig. 2.6, but Stability can find the true solution after Markov time 4. In practice, Stability uses time intervals in logspace, for example from 0.1 to 100 for 100 time points.

2.5.2 Markov Dynamics introduced to InfoMap

Markov dynamics were also introduced to the InfoMap method in [28]. InfoMap is a one-step method too, and it works like taking a snapshot of a network at Markov time 1. Using the Markov dynamics and Markov sweeping, we can find different community structures at different times.

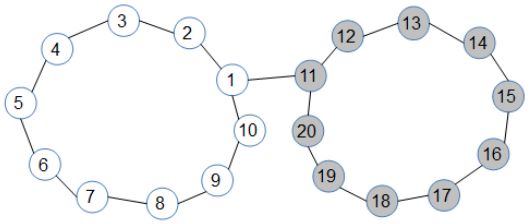
Markov dynamics can be introduced to the InfoMap equation in two easy steps. Unlike the original InfoMap method, time-dependent leaving probabilities q_α are used for each community α. Thus we need a time-dependent transition matrix, $M(t)$, of the network:

$$q_\alpha(t) = \sum_{i\in\alpha}\sum_{j\notin\alpha} p_i M_{ij}(t) \tag{2.5.7}$$

Secondly, we need to apply Markov sweeping to the original Map equation by scanning through all times. For a particular partition, we need to compute the InfoMap equation for all times up to time t, and choose the minimum value. Using the Markov dynamics as a zooming lens, InfoMap finds the communities at multiple scales.

Time interval for Stability analysis
100 Markov Times Between 10^{-1} and 10^{2}
time=logspace(-1,2,100);

0.1000	1.0723	10.7227
0.1072	1.1498	11.4976
0.1150	1.2328	12.3285
0.1233	1.3219	13.2194
0.1322	1.4175	14.1747
0.1417	1.5199	15.1991
0.1520	1.6298	16.2975
0.1630	1.7475	17.4753
0.1748	1.8738	18.7382
0.1874	2.0092	20.0923
0.2009	2.1544	21.5443
0.2154	2.3101	23.1013
0.2310	2.4771	24.7708
0.2477	2.6561	26.5609
0.2656	2.8480	28.4804
0.2848	3.0539	30.5386
0.3054	3.2745	32.7455
0.3275	3.5112	35.1119
0.3511	3.7649	37.6494
0.3765	4.0370	40.3702
0.4037	4.3288	43.2876
0.4329	4.6416	46.4159
0.4642	4.9770	49.7702
0.4977	5.3367	53.3670
0.5337	5.7224	57.2237
0.5722	6.1359	61.3591
0.6136	6.5793	65.7933
0.6579	7.0548	70.5480
0.7055	7.5646	75.6463
0.7565	8.1113	81.1131
0.8111	8.6975	86.9749
0.8697	9.3260	93.2603
0.9326	10.0000	100.0000
1.0000		



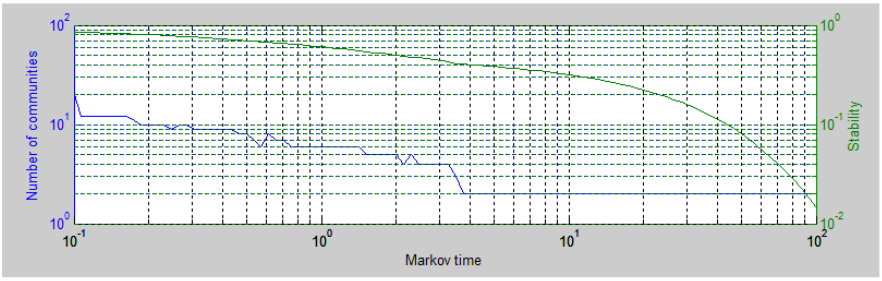


Figure 2.6: Stability can find communities in this network after Markov time 4. At left, we can see the all time points in logspace that Stability runs through. At the bottom, we can see the output of the Stability program [8]. The lower line shows the number of communities found at each Markov time, and the higher line shows the value of the Stability function for that Markov time.

2.6 Project Goals

Now that we have reviewed existing methods for community detection, we are able to state our project goals. To summarize, the existing methods we reviewed are as follows:

- InfoMap is one of the best performing community detection algorithms developed by Rosvall et al. [25].
- Barahona et al. introduced Markov Dynamics to InfoMap in Matlab for undirected networks [28].
- In separate work, Rosvall et al. modified InfoMap to find overlapping communities using the compression of flow [9].

In the remainder of this work, we describe how we have synthesized two existing methods:

- We introduced Markov Dynamics to InfoMap for directed networks in its original language C++.
- We modified InfoMap to detect overlapping communities using the compression of flow, similar to Rosvall et al.'s approach mentioned above, with small differences.

Chapter 3

Methods

In this section, we explain the methods and algorithms we are going to use for the multiscale community detection in complex networks. First, we will begin by explaining the InfoMap method and its optimization algorithm to find communities in complex networks. Then we will explain how to implement Markov dynamics on top of InfoMap to find communities at multiscales. Lastly, we will discuss how to modify InfoMap to find overlapping communities.

3.1 InfoMap

InfoMap finds the communities of a directed and weighted network by compressing the probability flows. We already gave the overview of InfoMap in Section 2.4.2. Now, we will explain the details of InfoMap in 4 steps:

- First, we explain how to compute the ergodic node visit frequencies in a network, the main element for the InfoMap equation.
- Second, we explain the InfoMap equation, and the delta InfoMap equation, which is important for algorithm efficiency(speed).
- Then, we explain the core of the optimization algorithm, which is almost similar to Blondel's algorithm [2].
- At last, we explain Rosvall's improvements to Blondel's algorithm.

3.1.1 Ergodic node visit frequency

If a random walker surfs on the nodes of a directed-weighted graph according to the weighted probabilities infinitely, then the time spent on each node will become stable. The resulting frequencies are called ergodic (steady-state) node visit frequencies. First, we will explain related concepts from Shannon & Weaver's book [29] as depicted in Fig. 3.1. A stochastic process is defined as "a system which produces a sequence of symbols according to certain probabilities" [29]. A Markov process is a special kind of stochastic process where the probabilities depend on the previous states (or events). Finally, an ergodic process is a special kind of Markov process that produces a sequence of symbols that has a statistical homogenity. Any reasonably large sample would be representative of the whole.

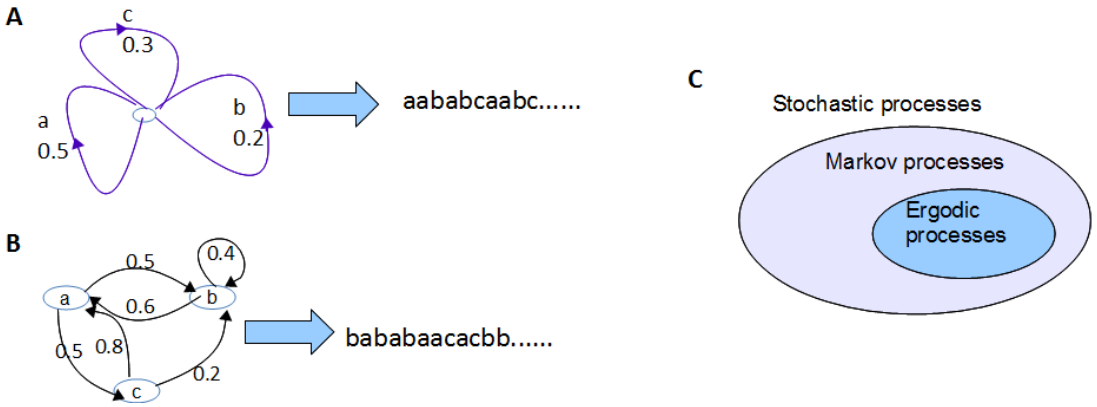


Figure 3.1: (A) A stochastic process, where the probabilities do not depend on previous state. (B) A Markov process, where the probabilities depend on previous state. (C) An ergodic process, which is a special kind of Markov process that produces a sequence of symbols where any sample of sufficient size has the statistical properties of the whole.

Before explaining the computation of ergodic node visit frequency, we will discuss the same phenomena from three different viewpoints as described in [34]. The ergodic node visit frequency can also be viewed as the importance or rank of each node. First, from a dynamical system viewpoint, assume that the importance of each node is the same at the beginning. At each iteration, each node gives all of its importance to its neighboring nodes according to the outgoing weights. Then the importance of each node will become stable and will not change after some iteration which is called equilibrium value. Secondly, from a linear algebra viewpoint, the directed-weighted graph can be described as an adjacency matrix where each column stores the probabilities of outgoing edges; thus the sum of each column is 1. The relative importance of each node can be solved by

finding the unique eigenvector corresponding to eigenvalue 1. Finally, from a probabilistic viewpoint, the relative importance of each node can be viewed as the probability of a random walker spending time at that node at stationarity. As a result, the equilibrium value, the eigenvector and the stationary distribution concepts are explaining the same phenomena from different views.

Infomap uses the Power Method Convergence Theorem to compute the ergodic node visit frequencies of a network:

Theorem 3.1.1 (Power Method Convergence Theorem) *Let M be a positive column-stochastic*[1] *nxn square matrix, and p* be its probabilistic eigenvector corresponding to eigenvalue 1 (stationary distribution). Let p be the column vector with all entries equal to 1/n (initial distribution). Then, the sequence p, Mp,* M^2p, M^3p, ..., M^kp *converges to the vector p*.*

Assume that we have a simple graph with four nodes as shown in Fig. 3.2A. At the beginning, the importance of each node or the initial distribution, p, is the same: 1/4. During the first iteration, each node distributes all of its importance, p, to the neighboring nodes according to the outgoing weights. After the first iteration, the new importance of each node will be the sum of its incoming flow. Because of that, the importance of each node depends only on the connectivity of the network, not the initial distribution. These calculations can be accomplished by multiplying the transition matrix of the network, T, with the initial distribution, p, as shown in Fig. 3.2B. We continue these iterations until the inflow of each node will become equal to the outflow of that node. Then the system will reach equilibrium, and the distribution will reach stationarity and will not change anymore. This particular network reached equilibrium after 15 iterations.

The power method uses an initial distribution to find the eigenvectors of a matrix or stationary distribution of a flow. Because of that, it starts with initial distribution p = 1/n. In fact, the stationary distribution is independent of initial distribution: it only depends on the connectivity patterns within the network.

[1] *A matrix is positive column-stochastic, if all entries are positive probabilities* $<= 1$*, and the sum of each column is 1.*

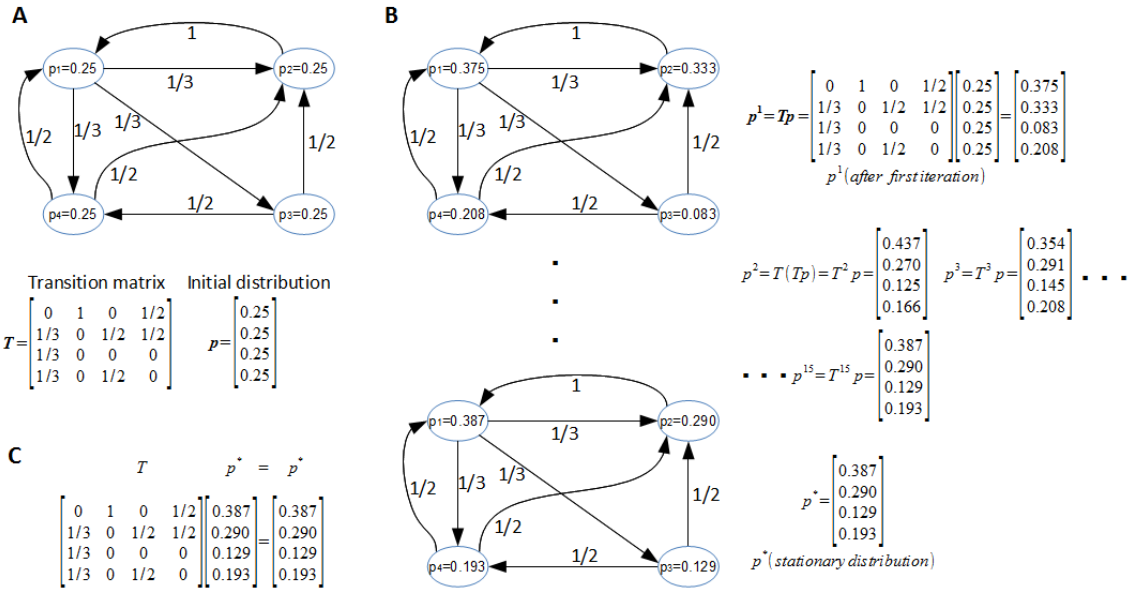


Figure 3.2: (A) A simple network with four nodes with its transition matrix T, and initial distribution p. (B) Power method applied to this network to find the ergodic node visit frequency, p*, or stationary distribution. (C) p* is also the eigenvector of the transition matrix T.

Although our solution may be correct for this simple network, we have to be sure of the correctness of the solution for complex real life networks. First, does a stationary distribution or an eigenvector for the transition matrix exist? Secondly, if there is a stationary distribution or an eigenvector, is it unique? The Perron-Frobenious theorem answers both of these questions:

Theorem 3.1.2 (Perron-Frobenious Theorem) *Let M be a positive column-stochastic irreducible nxn square matrix, then:*

- *1 is an eigenvalue of multiplicity one.*
- *1 is the largest eigenvalue, and the absolute value of all other eigenvalues are smaller than 1.*
- *the eigenvector corresponding to eigenvalue 1 has all entries positive.*
- *for the eigenvalue 1 there exists a unique eigenvector with the sum of its entries equal to 1.*

According to the Perron-Frobenious theorem, an irreducible and aperiodic Markov-chain has a unique stationary distribution. Math has exact solutions for perfect networks. In this case, the network should be irreducible, which means that it should be possible to go from any node to any other node in finite time, and aperiodic, which means a random walker should not be able to

visit any node periodically. Periodic visitation means that the same node is visited deterministically after a given number of steps. Nodes may be revisited in aperiodic networks but not at predictably regular times. But real-world networks are not perfect, there may be dangling nodes, dead-end loops or disconnected components as shown in Fig. 3.3. Because of these issues, many large real-world networks are not irreducible and not aperiodic. Thus we cannot find their unique stationary distribution using the Power method.

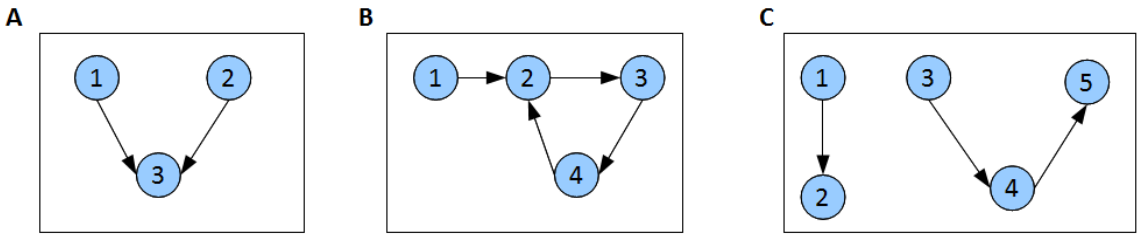


Figure 3.3: (A) Dangling nodes (nodes with no outgoing edges). (B) Dead end loops. (C) Disconnected components.

Google founders Page & Brin introduced teleportation probability to close the gap between theory and application, which is the original idea behind the Google's PageRank algorithm [4]. Assume the actual transition matrix of a real-world network (T) is not irreducible and not aperiodic. They introduced a teleportation matrix B that connects any node to any other one with non-zero probability 1/n. Thus B is definitely irreducible and aperiodic. If we combine these two transition matrices with some teleportation probability λ (or damping factor in PageRank lingo), then the result, $(1 - \lambda)T + \lambda P$ will become irreducible and aperiodic, and still it is very similar to the actual network. Therefore its stationary distribution can be considered as the real network's stationary distribution. The typical value for teleportation probability is 0.15, and InfoMap uses 0.15 too [4, 25]. 0.15 is chosen by experimental results, but there are some interesting research about how to choose the teleportation probability [15, 3], which is out of our scope.

Normally, a random walker can only go to neighboring nodes according to outgoing probabilities. When we introduce teleportation, the random walker becomes a "random surfer". With 0.85 probability it goes to neighboring nodes according to outgoing weights, and with 0.15 probability it goes to any node within the network randomly [25]. Thus, the network becomes irreducible and aperiodic. Dangling nodes, dead-end loops or disconnected components cannot trap the "random surfer".

Our simple network in Fig. 3.4 is already irreducible and aperiodic, so does not need teleportation. But again we demonstrate the computation of ergodic node visit frequency using the Power method with teleportation for this network in Fig. 3.4. Using the power method and teleportation, we can compute the unique eigenvector(stationary distribution) of a sparse matrix with millions of nodes, in seconds or minutes.

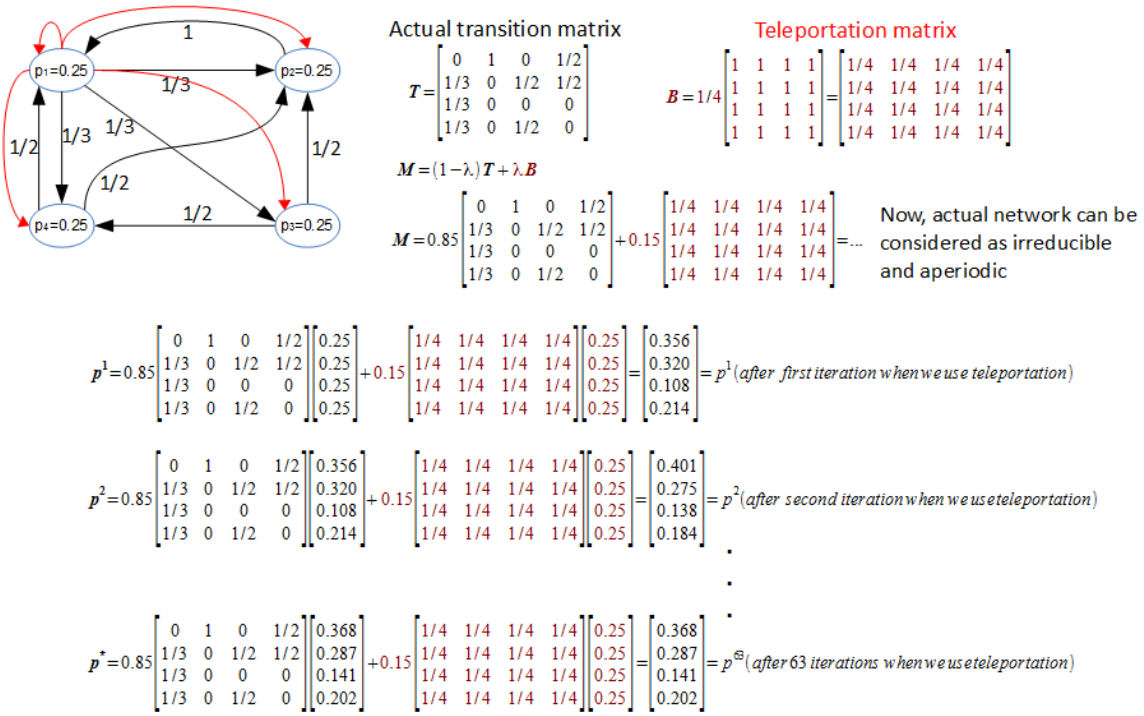


Figure 3.4: Computing the stationary distribution using teleportation probability $\lambda = 0.15$. In the figure, we show the actual flow with black links, and teleportation flow with red links. We show the red links for only one node for the sake of simplicity, but each node has the same links in computations. At each iteration, each node distributes 0.85 fraction of its importance to the neighboring nodes according to weights, and 0.15 fraction to all nodes within the network, including itself.

3.1.2 InfoMap Equation

In this section, we will explain the details of the InfoMap equation. Shannon's famous source coding theorem sets the compression limit for the data generated by a stochastic process. If the data is generated by n different codewords that correspond to n states of a random variable X with frequencies p_i, then the average length of a codeword cannot be compressed less than the entropy of random variable X, $H(X) = \sum_{i=1}^{n} p_i \log(p_i)$ [29, 25]. A random variable is a variable that

takes different values according to certain probabilities, such as a coin or a dice. As we can see in Fig. 3.5, the entropy of an unfair coin is less than the entropy of a fair coin; therefore, the trajectory of an unfair coin can be compressed better than a fair coin because there are more regularities in its trajectory.

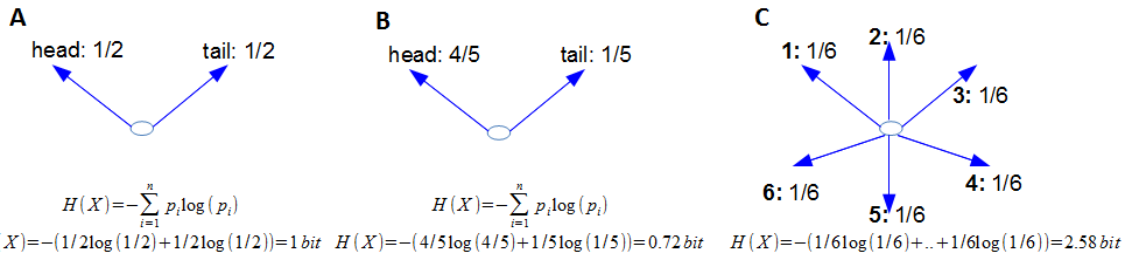


Figure 3.5: (A) A fair coin which has equal probability of being tail or head. (B) An unfair coin with probability of being head=0.8 and tail=0.2. (C) A fair dice with equal probability of 1/6 for all faces.

Infomap finds the communities of a network by compressing the trajectory of a random walker on that network. Following InfoMap notation, we will use α, β to enumerate nodes, and i, j to enumerate modules. Assume we have a network with n=15 nodes which is partitioned into m=3 modules as shown in Fig. 3.6A, and their ergodic node visit frequencies, p_α, are shown in Fig. 3.6B with black arrows. If we give a unique codeword for each node in the network, we may compress the trajectory but we cannot find communities, therefore InfoMap uses a two-level coding. There is one index codebook to encode the movement between modules, and m module codebooks to encode the movements within each module. Codewords in each module are unique, but the same codewords can be reused within different modules. Because of that, we need an exit codeword (the red arrows), q_i, for each community, to indicate that we are exiting the community, and we use the index codebook while entering a new module. Thus there will be no conflict due to codeword reusage. We already computed the ergodic node visit frequencies, p_α, for each node. The exit probability, q_i, for each community i can be computed as

$$q_i = \lambda \frac{n - n_i}{n} \sum_{\alpha \in i} p_\alpha + (1 - \lambda) \sum_{\alpha \in i} \sum_{\beta \notin i} p_\alpha w_{\alpha\beta} \tag{3.1.1}$$

where the first term is the exit flow from the module due to teleportation ($\lambda = 0.15$), and the second term is the exit flow from the boundary nodes according to their outgoing weights $w_{\alpha\beta}$.

The exit code is also considered as an artificial codeword within the module codebook, thus the sum of ergodic node visit frequencies in module i is:

$$p^i = q_i + \sum_{\alpha \in i} p_\alpha \tag{3.1.2}$$

We can compute the entropy of movement within module i, H(P^i), using the normalized value of ergodic node visit frequencies and exit probability:

$$H(P^i) = (\frac{q_i}{p^i}) log(\frac{q_i}{p^i}) + \sum_{\alpha \in i} (\frac{p_\alpha}{p^i}) log(\frac{p_\alpha}{p^i}) \tag{3.1.3}$$

An interesting point is that in an ergodic flow, the inflow equals the outflow, because of that InfoMap uses the exit probability, q_i, of each module also as the frequency of that module in the index codebook, the central codebook in Fig. 3.6B. Thus, we can compute the entropy of movement between m modules, H(Q), using the normalized value of exit frequencies, q_i, which can be acquired by dividing the q_i with the sum of all exit frequencies $q = \sum_{i=1}^{m} q_i$.

$$H(Q) = \sum_{i=1}^{m} (\frac{q_i}{q}) log(\frac{q_i}{q}) \tag{3.1.4}$$

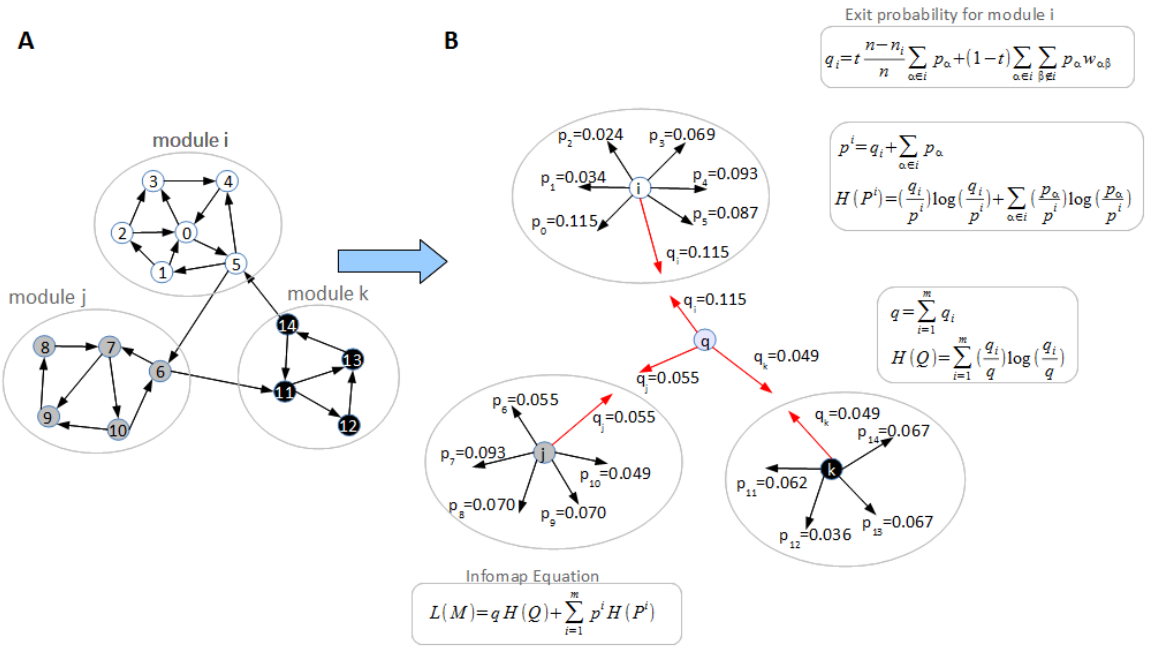


Figure 3.6: The partition of the network in (A) can also be viewed as there is a random variable within each module, and one random variable between modules as shown in (B). So we can compute the codeLength of the network using ergodic node visit frequencies.

Now, we have the entropies for m module codebooks $H(P^i)$, and one index codebook $H(Q)$. The entropy of movement between the modules and the entropy of movement within modules are against each other, and cannot be reduced at the same time. The final Infomap equation, L(M), balances the entropy of movement between the modules and within the modules using the Minimum Description Length principle. Also, the entropies are weighted according to the rates of use (usage time of each codebook). The entropy of movement between modules, H(Q), is weighted with q, and the entropy of movement within each module i, $H(P^i)$, is weighted with p^i. Among many possible partitions, the partition that minimizes the InfoMap equation is the best one.

$$L(M) = qH(Q) + \sum_{i=1}^{m} p^i H(P^i) \tag{3.1.5}$$

When we do the simplifications as shown in Fig. 3.7, we can see that InfoMap equation is only a function of ergodic node visit frequencies p_α and exit probabilities q_i [25]. Furthermore, $\sum_{i=1}^{n} p_\alpha log(p_\alpha)$ is independent of partitioning. Therefore, the optimization algorithm only needs to keep track of changes in q_i and $\sum_{\alpha \in i} p_\alpha$.

$$L(M)=q\,H(Q)+\sum_{i=1}^{m} p^i H(P^i)$$

$$q=\sum_{i=1}^{m} q_i \qquad\qquad p^i=q_i+\sum_{\alpha\in i} p_\alpha$$

$$H(Q)=-\sum_{i=1}^{m}\frac{q_i}{q}\log\left(\frac{q_i}{q}\right) \qquad\qquad H(P^i)=-\frac{q_i}{p^i}\log\left(\frac{q_i}{p^i}\right)-\sum_{\alpha\in i}\frac{p_\alpha}{p^i}\log\left(\frac{p_\alpha}{p^i}\right)$$

$$L(M)=-q\sum_{i=1}^{m}\frac{q_i}{q}\log\left(\frac{q_i}{q}\right)-\sum_{i=1}^{m}p^i\left(\frac{q_i}{p^i}\log\left(\frac{q_i}{p^i}\right)+\sum_{\alpha\in i}\frac{p_\alpha}{p^i}\log\left(\frac{p_\alpha}{p^i}\right)\right)$$

$$L(M)=-\sum_{i=1}^{m} q_i\log\left(\frac{q_i}{q}\right)-\sum_{i=1}^{m}\left(q_i\log\left(\frac{q_i}{p^i}\right)+\sum_{\alpha\in i} p_\alpha\log\left(\frac{p_\alpha}{p^i}\right)\right)$$

$$L(M)=-\sum_{i=1}^{m} q_i\log(q_i)+\sum_{i=1}^{m} q_i\log(q)-\sum_{i=1}^{m}\left(q_i\log(q_i)-q_i\log(p^i)+\sum_{\alpha\in i} p_\alpha\log(p_\alpha)-\sum_{\alpha\in i} p_\alpha\log(p^i)\right)$$

$$L(M)=-\sum_{i=1}^{m} q_i\log(q_i)+\sum_{i=1}^{m} q_i\log(q)-\sum_{i=1}^{m}\left(q_i\log(q_i)+\sum_{\alpha\in i} p_\alpha\log(p_\alpha)-(q_i+\sum_{\alpha\in i} p_\alpha)\log(p^i)\right)$$

$$L(M)=-\sum_{i=1}^{m} q_i\log(q_i)+\sum_{i=1}^{m} q_i\log(q)-\sum_{i=1}^{m} q_i\log(q_i)-\sum_{i=1}^{m}\sum_{\alpha\in i} p_\alpha\log(p_\alpha)+\sum_{i=1}^{m}(q_i+\sum_{\alpha\in i} p_\alpha)\log(p^i)$$

$$L(M)=\left(\sum_{i=1}^{m} q_i\right)\log\left(\sum_{i=1}^{m} q_i\right)-2\sum_{i=1}^{m} q_i\log(q_i)-\sum_{\alpha=1}^{n} p_\alpha\log(p_\alpha)+\sum_{i=1}^{m}\left(q_i+\sum_{\alpha\in i} p_\alpha\right)\log\left(q_i+\sum_{\alpha\in i} p_\alpha\right)$$

Figure 3.7: Simplification steps of InfoMap equation. It helps in the implementation of the codeLength and ΔcodeLength computations.

Of course, the "simplified" equation looks more complex! But it is simplification of the computation needed, not of the mathematical expression.

3.1.3 InfoMap optimization algorithm

In this section, we will explain the optimization algorithm that finds the communities of a network by minimizing the InfoMap equation. We will illustrate how InfoMap works on a simple network with 15 nodes and 25 arcs(directed edges) as depicted in Fig. 3.8. The core of the optimization algorithm is very similar to the Louvain method (aka Blondel's algorithm) as explained in [2]. In brief, the optimization algorithm has two phases, which are repeated iteratively. In the first phase, each node has its own module, so the number of nodes and the number of modules are same. In a for loop, the algorithm chooses a node in random sequential order[2], and computes the gain in codeLength (ΔcodeLength) by removing the node from its own module and joining one

[2]Random sequential order means a node is chosen randomly but only once in a loop.

of the neighboring modules. Then the node is moved to the neighboring module which results in the highest positive gain in codeLength. If there is no positive gain, the node stays in the same module. This iteration continues until there is no movement between modules; therefore a node can be considered to move between modules several times. In the second phase, the modules are converted to nodes and a new network constructed at a higher level. The size, links, exit probabilities and other features of the new nodes are derived from the old modules, and the first phase of the algorithm applied to the new network. If we call these two phases one iteration, these iterations continue until there is no positive gain in codeLength.

Initialization: The InfoMap algorithm takes a network(graph) as an input, in Pajek or LinkList format as shown in Fig. 3.8A. As a first step, it normalizes the outlink of each node, so the transition matrix of the graph will become column-stochastic (see Fig. 3.8B) (the sum of each node's outgoing links equals 1). Then InfoMap computes the ergodic node visit frequency of each node using the power method as described in section 3.1.1 (see Fig. 3.8C). After this computation, the sum of all ergodic node visit frequencies equals 1.

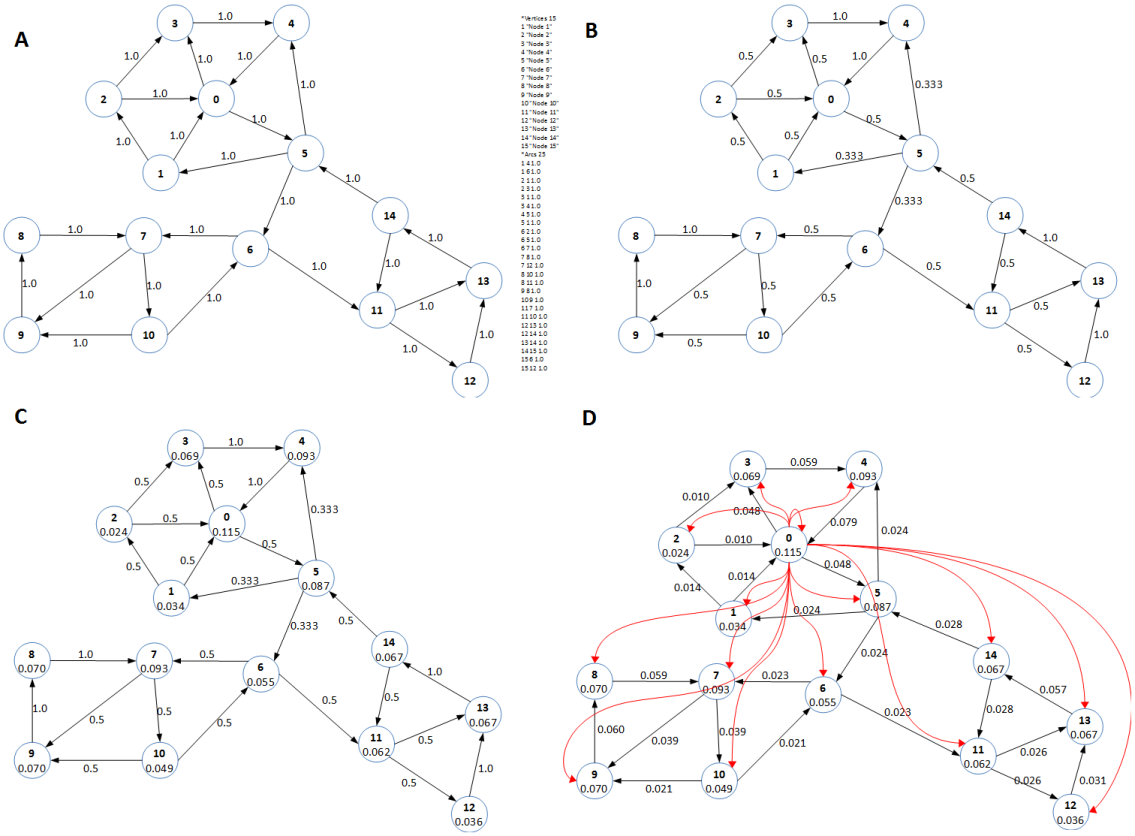


Figure 3.8: Initialization steps of InfoMap optimization algorithm. (A) The input graph to InfoMap. (B) Outlinks are normalized. (C) Ergodic node visit frequency of each node is computed. (D) The weight of each link is updated to represent the flow.

Now the size of each node equals the ergodic node visit frequency of that node. In the next step, InfoMap updates the weight of each outgoing link to represent the flow. In Fig. 3.8D, the black links correspond to real links between a node and its neighbors, and the red links represent the teleportation links to all nodes including a link to itself. We just displayed the teleportation links of node 0 for the sake of simplicity, but actually all nodes have teleportation links to all other nodes. InfoMap uses a well-known teleportation probability (a.k.a. damping factor) of 0.15. Each node distributes 0.85 of its size to the neighbors through outgoing links, and the remaining 0.15 of its size to all network via teleportation links. For example, in Fig. 3.8D, the size of node 0 is 0.115 and it has two outgoing real (black) links, and 15 outgoing teleportation (red) links. Therefore, the weight of each outgoing link will become $(0.115 * 0.85)/2 = 0.048$, and the weight of each outgoing teleportation link will become $(0.115 * 0.15)/15 = 0.00115$.

Exit probabilities: At the initiation, InfoMap computes the exit probabilities of each module where each node is also a module. The exit probability is the probability of a random walker to exit from the module at any time, and the exit probability q_i for each module i with n_i nodes is

$$q_i = \lambda \frac{n - n_i}{n} \sum_{\alpha \in i} p_\alpha + (1 - \lambda) \sum_{\alpha \in i} \sum_{\beta \notin i} p_\alpha w_{\alpha\beta} \tag{3.1.6}$$

The first term finds the size of teleportation flow exiting from the module, since every node distributes $\lambda = 0.15$ fraction of its size to all nodes, it is weighted with the fraction of teleportation links going outside of the module $\frac{n-n_i}{n}$. The second term finds the size of real flow exiting from the module. Each node distributes $(1 - \lambda) = 0.85$ of its size to neighboring nodes proportionally with the weights of outgoing links $\sum_{\alpha \in i} \sum_{\beta \notin i} w_{\alpha\beta}$ (see Fig. 3.9).

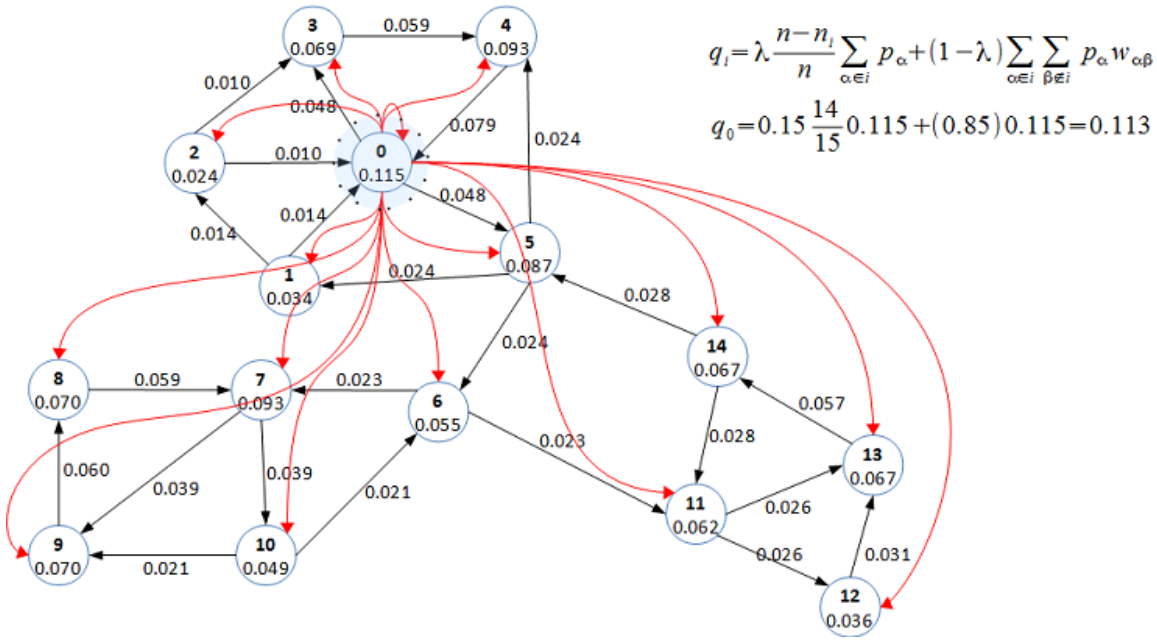
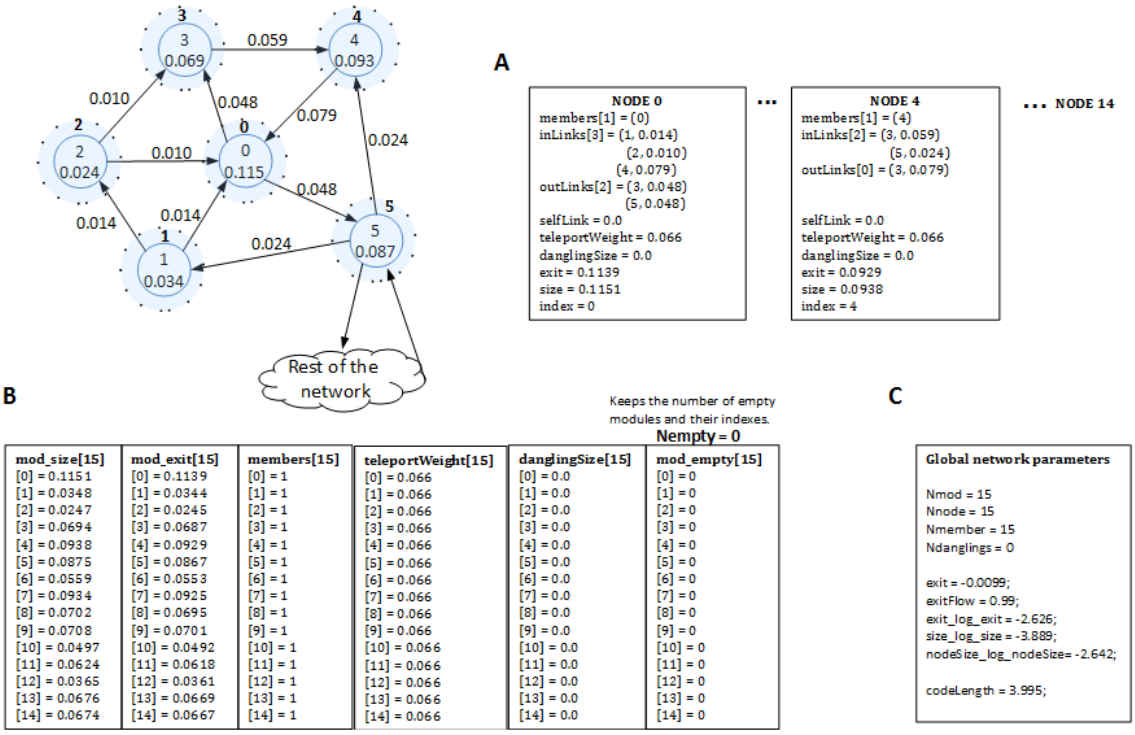


Figure 3.9: Computation of exit flow for module 0, q_0. Black arrows are real links and red arrows represent the teleportation flow. Actually all nodes have teleportation flow, but we only display for node 0, for the sake of simplicity.

Data structure: the InfoMap algorithm uses three types of data structure: nodes, modules and global network parameters as shown in Fig. 3.10. The similarity between node attributes and module attributes is easily recognizable. In fact, the node objects and module vectors store the same values at the beginning of each network level. During the process of movement of nodes be-

tween modules, the attributes of node objects do not change (except for the index), but the module vectors and global network parameters are modified to keep track of updated values. Therefore Δ codeLength can be easily computed.



mod_size[15]	mod_exit[15]	members[15]	teleportWeight[15]	danglingSize[15]	mod_empty[15]
[0] = 0.1151	[0] = 0.1139	[0] = 1	[0] = 0.066	[0] = 0.0	[0] = 0
[1] = 0.0348	[1] = 0.0344	[1] = 1	[1] = 0.066	[1] = 0.0	[1] = 0
[2] = 0.0247	[2] = 0.0245	[2] = 1	[2] = 0.066	[2] = 0.0	[2] = 0
[3] = 0.0694	[3] = 0.0687	[3] = 1	[3] = 0.066	[3] = 0.0	[3] = 0
[4] = 0.0938	[4] = 0.0929	[4] = 1	[4] = 0.066	[4] = 0.0	[4] = 0
[5] = 0.0875	[5] = 0.0867	[5] = 1	[5] = 0.066	[5] = 0.0	[5] = 0
[6] = 0.0559	[6] = 0.0553	[6] = 1	[6] = 0.066	[6] = 0.0	[6] = 0
[7] = 0.0934	[7] = 0.0925	[7] = 1	[7] = 0.066	[7] = 0.0	[7] = 0
[8] = 0.0702	[8] = 0.0695	[8] = 1	[8] = 0.066	[8] = 0.0	[8] = 0
[9] = 0.0708	[9] = 0.0701	[9] = 1	[9] = 0.066	[9] = 0.0	[9] = 0
[10] = 0.0497	[10] = 0.0492	[10] = 1	[10] = 0.066	[10] = 0.0	[10] = 0
[11] = 0.0624	[11] = 0.0618	[11] = 1	[11] = 0.066	[11] = 0.0	[11] = 0
[12] = 0.0365	[12] = 0.0361	[12] = 1	[12] = 0.066	[12] = 0.0	[12] = 0
[13] = 0.0676	[13] = 0.0669	[13] = 1	[13] = 0.066	[13] = 0.0	[13] = 0
[14] = 0.0674	[14] = 0.0667	[14] = 1	[14] = 0.066	[14] = 0.0	[14] = 0

Figure 3.10: InfoMap implementation uses three types of data structures: (A) Each node is represented as an object. (B) Module attributes are stored in several vectors. (C) Global network parameters are stored in single parameters.

- Nodes: Each node is represented as an object as shown in Fig. 3.10A. This object contains the lists of incoming links, outgoing links and members. Although a member list looks unnecessary for a node, later stages of the recursive algorithm converts modules to nodes at a new network level and reruns the algorithm. Thus nodes in higher levels may have many original nodes as members from a previous level. Additionally, a node object contains the ergodic node visit frequency (size), the exit probability and the id (index) of the module it belongs to. If the node is a dangling node, the size of the node is also copied to the danglingSize attribute to take care of dangling nodes in the algorithm, because dangling nodes don't have any outgoing links and they distribute all of their size to the whole network via teleportation links. If the node has a self link, then the weight of the self link is stored in this attribute, not in the inLinks or outLinks. Lastly, there is a teleportWeight for each node, which is used to

determine the fraction of nodes in that node/module. At the initiation of the algorithm, the teleportWeight of all nodes are the same: 1/n, but later the modules become nodes and the teleportWeight of a node is proportional to the number of original nodes inside that node.

- Modules: Modules are not represented with an object, but their attributes are stored in several vectors. The node attributes and module attributes are similar, and it is useful while converting modules to nodes at a new level.
- Global network parameters: these store the network level parameters. Nmod stores the number of original nodes, which never changes, whereas Nmember stores the number of modules and nodes at that network level, which changes at every new network level. Additionally it stores the parameters to compute the codeLength.

CodeLength computation: After the initialization, InfoMap computes the initial codeLength of the partition using the global network parameters as shown in Fig. 3.11. Global network parameters can be acquired from the module vectors. $exitFlow$ is the sum of all exit probabilities, and $exit$ is its plogp value. $size\_log\_size$ and $exit\_log\_exit$ are the sum of all plogp values of size and exit probabilities respectively. $nodeSize\_log\_nodeSize$ is the sum of plogp value of initial ergodic node visit frequencies which never changes during the process. The initial codeLength for this 15 node network is 3.995 bits, very close to $\log_2(15)$.

$$L(M)=\underbrace{\left(\sum_{i=1}^{m} q_i\right)\log\overbrace{\left(\sum_{i=1}^{m} q_i\right)}^{\texttt{exitFlow}}}_{\texttt{exit}}-\underbrace{2\sum_{i=1}^{m} q_i\log(q_i)}_{\texttt{exit\_log\_exit}}-\underbrace{\sum_{\alpha=1}^{n} p_\alpha\log(p_\alpha)}_{\texttt{nodeSize\_log\_nodeSize}}+\underbrace{\sum_{i=1}^{m}\left(q_i+\sum_{\alpha\in i} p_\alpha\right)\log\left(q_i+\sum_{\alpha\in i} p_\alpha\right)}_{\texttt{size\_log\_size}}$$

```
for(int i=0;i<Nmod;i++){
  exit_log_exit += plogp(node[i]->exit);
  size_log_size += plogp(node[i]->exit + node[i]->size);
  exitFlow += node[i]->exit;
}
exit = plogp(exitFlow);

codeLength = exit - 2.0*exit_log_exit + size_log_size - nodeSize_log_nodeSize;
```

Figure 3.11: Computation of the codeLength.

Core of the optimization algorithm: After the initialization, the core of the optimization algorithm finds the modules of the network in two phases. In the first phase, the algorithm picks a random node in a loop, and finds the flow between this node and its neighboring modules

(including its own module and possibility of moving to an empty module, if not already alone), and stores this in the flowNtoM vector. Using this vector, InfoMap easily computes the gain in codeLength (ΔcodeLength) by removing this node from its own module and adding it to another module without recomputing the codeLength of the whole network. This optimization has a very significant effect on the efficiency and speed of the algorithm.

As depicted in Fig. 3.12C-D, when a node changes its module, we can subtract its size, danglingSize, teleportWeight and memberSize from the oldModule, and add them to the respective attributes of newModule by simple arithmetic operations. But updating exit probabilities needs more care: when we remove node[i] from a module, the exit probability of oldModule decreases by its contribution $node[i] \rightarrow exit - outFlowOldM$ and increases by $inFlowOldM$ since the node is going out of module. In the same manner when we add node[i] to a module, the exit probability of newModule increases by its contribution $node[i] \rightarrow exit - outFlowNewM$ and decreases by $inFlowNewM$ since the node will be inside the module. Therefore we can compute the ΔcodeLength as shown in Fig. 3.12E.

InfoMap chooses a node in random sequential order and makes the best move. The first phase of the core algorithm continues until there is no movement between modules. The InfoMap implementation also uses a small threshold value to prevent an infinite loop. If the gain in codeLength is smaller than the treshold value then the first phase stops even there is movement between modules.

In the second phase of the algorithm, modules are converted into nodes, and a new network generated at a higher level. The properties of the new nodes (size, danglingSize, teleportWeight, exitProbability and members) are copied from the modules of the previous level. The outLinks/inLinks between the new nodes are calculated as the sum of outLinks/inLinks between the nodes of corresponding modules at the previous level. These two phases forms an iteration, and continue until there is no improvement on the codeLength. The only difference between the Louvain method and the InfoMap's implementation is about the order of nodes in the loop; the Louvain method picks nodes in the given order whereas InfoMap picks them in random order.

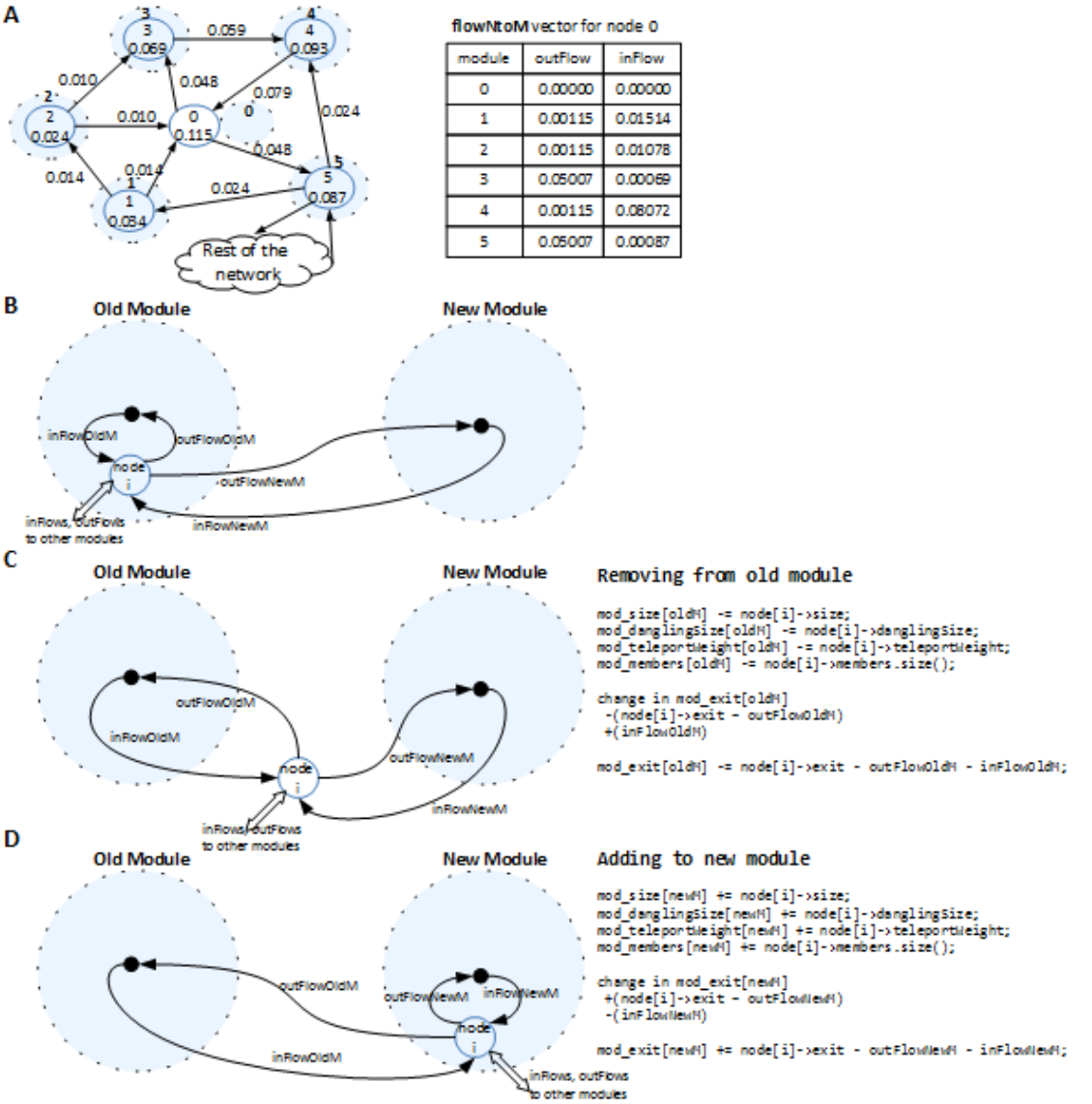


Figure 3.12: Part of the efficiency of the algorithm comes from the fact that ΔcodeLength can be computed easily when a node changes its module.

```
void move(bool &moved){

      moved = false;
      vector<pair<int,pair<double,double>>> flowNtoM(Nnode); //[module, inFlow, outFlow]
      // Pick nodes in random order
      for(int k=0;k<Nnode;k++){

            fill_flowNtoM(node[k]);
            Fill the flowNtoM vector for node[k]
            - using all outlinks/inLinks of node[k];
            - add weights to flowNtoM[][] from teleportation and dangling nodes;
            - calculate flow to/from own module of node[k];
            - calculate option to move to empty module (if not already alone);

            find_the_best_move(node[k]);
            // find the best move that minimizes the description length
            double best_delta_codeLength = 0.0; int bestM = oldM;
            for (int j=0; j<flowNtoM.size; j++) {

                  compute delta_codeLength for moving node[k] to module[j];
                  if(delta_codeLength < best_delta_codeLength){
                        bestM = newM;
                        best_delta_codeLength = delta_codeLength;
                  }
            }

            make_the_best_move(node[k]);
            // make best possible move, by updating terms
            if(bestM != oldM){
                  //update module vectors for old and new module(mod_size, mod_exit ….)
                  //update the id of node[k] from oldModule to newModule
                  //update terms in map equation & codeLength
                  moved = true;
            }
      }
}
```

Figure 3.13: First phase of the InfoMap optimization algorithm continues until there is no movement between modules. (B) The second phase rebuilts a new network where its nodes are the modules of previous level. These two phases are applied to the network until there is no improvement in codeLength.

```
void coreAlgorithm(){

   double oldCodeLength;
   do{
      oldCodeLength = codeLength;
      bool moved = true;
      int count = 0;

      while(moved){
           moved = false;
           inner_oldCodeLength = codeLength;

           move(moved); // call move function

         // to prevent infinite loop, use a treshold value
           if(codeLength - inner_oldCodeLength < 1.0e-10)
                moved = false;

         //recompute(tune) module attributes and map equation terms after 10 loops
           count++;
           if(count == 10){
                tune();
                count = 0;
           }
      }

      level(node,true);
      Convert modules to nodes, and construct a new network at a higher level
      - copy the size, danglingSize, teleportWeight, exitProbability & members
        of each module to the corresponding new node at the new level
      - create inLinks/outLinks between new nodes using the sum of inLinks/outLinks
        between the nodes of corresponding modules at the previous level.

   }while(oldCodeLength - codeLength >  1.0e-10);
}
```

Figure 3.14: The second phase of the optimization algorithm rebuilts a new network where its nodes are the modules of previous level. These two phases are applied to the network until there is no improvement in codeLength.

Improvements by Rosvall: After the first phase of the core algorithm; the modules are converted to nodes, a new network is rebuilt and the recursive algorithm reapplied. Once the network is rebuilt at a new level, the large number of nodes that are assigned to a module are forced to move together at later iterations. A move that looks optimal in the early iterations may have a negative effect at the later iterations or the final state, but there is no way to separate them again in this algorithm [23]. Because of that, Rosvall made extensions to improve the accuracy of the algorithm. These extensions break the modules of the final state of the core algorithm in two ways:

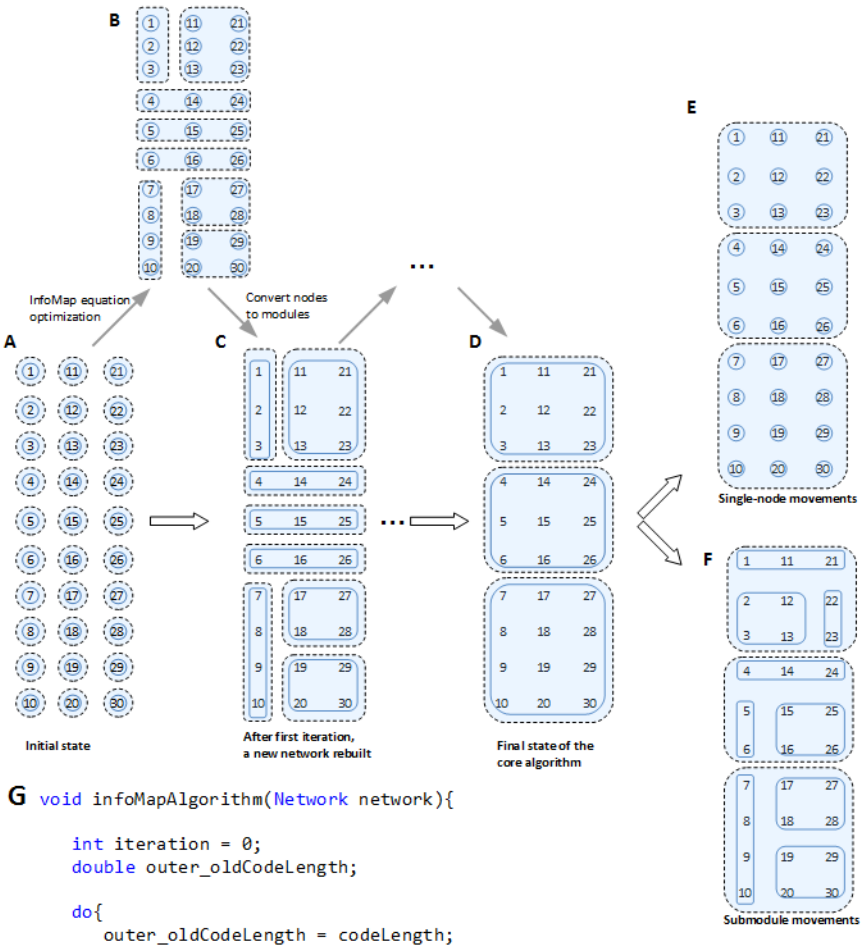
Single-node movements: As we know, the final state of the network shows the modules that may be acquired after several iterations. At each iteration, the modules are converted to nodes and the main algorithm reapplied. For single-node movements, the final state of the network is prepared as follows: each original single node has its own module and is also made directly a

member of the final modules by skipping the intermediate level modules. Then the core algorithm is applied to this network, so each single node can move freely between the modules of the final state as shown in Fig. 3.15E.

Submodule movements: First, each module of the final state is considered as a separate network and the main algorithm finds submodules of this network (module). After this process, each module may have one or more submodules. Then, the final state of the network is prepared as follows: each submodule is directly a member of the final module, and the core algorithm is applied to this network, so submodules can move freely between the modules of the final state as shown in Fig. 3.15F. Furthermore, submodule movements are applied recursively such as breaking submodules into subsubmodules, so on.

These two extensions are applied to the main algorithm in sequence until there is no improvement in codeLength. Single-node movements and submodule movements act like fine-tuning and coarse-tuning of the optimization algorithm respectively. The algorithm of these two extensions are shown in Fig. 3.15G and illustrated roughly in Fig. 3.15EF. For the sake of simplicity, the illustrations present the nodes without links.

Since this algorithm is stochastic, each run of the algorithm on a network may result in a (slightly) different partition and different codeLength. Because of that, InfoMap repeats the main algorithm "nTrial" times and saves the partition that results in the shortest codeLength. The "nTrial" parameter is given to the InfoMap program from the command-line. This ends the explanation of the InfoMap algorithm.




```
void infoMapAlgorithm(Network network){

    int iteration = 0;
    double outer_oldCodeLength;

    do{
       outer_oldCodeLength = codeLength;

       //Repeat the Rosvall's two extensions to the coreAlgorithm in sequence
       if((iteration > 0) && (iteration % 2 == 0) && (Nnode > 1)){
            Submodule movements
            - for each module of the final state, run the main infoMapAlgorithm to find submodules
            for(int i=0; i<network->numOfModules; i++){
                 infoMapAlgorithm(network->module[i])
            }
            - prepare the final state of the network:
            - where the submodules are directly member of the final modules without intermediate levels

       }else if(iteration > 0){
            Single-node movements
            - prepare the final state of the network:
            - where the original single-nodes are directly
              member of the final modules without intermediate levels
       }

       coreAlgorithm();  // call the coreAlgorithm (Blondel's)

       iteration++;
    }while(outer_oldCodeLength - codeLength > 1.0e-10);
}
```

Figure 3.15: Single-node and submodule movements. Dashed and continuous lines represent the boundaries of modules and nodes respectively. (A-D) Illustration of the core algorithm. (E-F) The illustration of Rosvall's extensions. (G) Rosvall's extensions to the core algorithm.

3.2 Markov Dynamics

Barahona et al. already introduced Markov dynamics to the InfoMap algorithm to find communities at multiscale using Matlab [28]. Following their idea, we implemented Markov dynamics to InfoMap in its original language C++. In this section, we will explain how we introduced Markov dynamics to the InfoMap algorithm to find communities at multiscale.

First, we get the adjacency matrix, A, of a directed network where each row i shows the weights of outgoing links for node i as shown in Fig. 3.16A. Then we generate the transition matrix M, by normalizing the outgoing links of each node, so the sum of each row becomes 1, because the matrix has to be row stochastic to describe a Markov chain. A random walker cannot get lost, at any node it can go to some another node, and the flow is conserved. If we use discrete-time Markov process, we can get the powers of this row-stochastic transition matrix and use it in InfoMap, because the transition matrix M and all of its powers are row-stochastic.

Since we are interested in continuous-time Markov process, we need to go from discrete-time to continuous-time Markov. For discrete-time, the transition matrix, M, and all of its powers are row stochastic, so there is no problem about the conservation of flow at discrete Markov times such as 2,3,4 etc. For continuous time Markov dynamics, we need to get the exponential of the transition matrix M, for Markov times such as 0.56 or 2.35. But the exponential of the transition matrix are not row stochastic, the sum of each row becomes greater than 1, and getting larger for higher Markov times. To overcome this problem, we need to subtract the identity matrix from the transition matrix, (M-I), to make the flow conserved [28]. Now the matrix exponential of (M-I) for all continuous Markov times are row stochastic[3], and the flow is conserved for all Markov times, as shown in Fig. 3.16B.

[3] Not a proof, but the intuition behind (M-I) is: if we have 1x1 matrix and its row-sum = 1, since $1^n = 1$, the powers of M are row-stochastic for discrete-time Markov process. For continuous-time Markov process: if its row-sum = 1, $e^1 > 1$ and it is not row-stochastic, but if its row-sum = 0 then $e^0 = 1$, because of that we are subtracting Identity matrix from M. Therefore matrix exponential of (M-I) becomes row-stochastic.

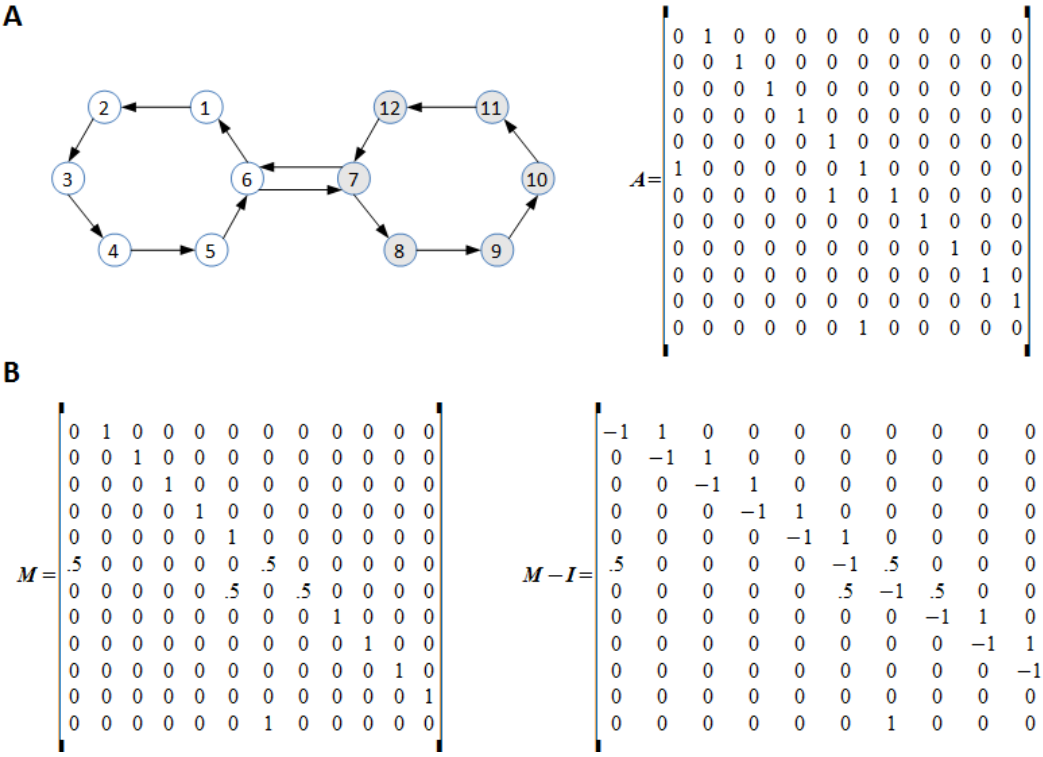


Figure 3.16: (A) The network and its adjacency graph, A. (B) The transition matrix of the network M, and (M-I).

Then we specify a time interval in log space to analyze the network, for example logspace (-1, 2, 100), 100 time points between 0.1 and 100. Since most real life phenomena such as sounds, earthquakes are measured in logarithmic scale, using log scale may be more meaningful than linear scale. For each time point, t, we compute the matrix exponential of the $(t * (M - I))$ using the scaling and squaring method. We have used the code explained in [19] with small changes in the parameters.

There are many methods to compute matrix exponentials, and they usually cannot compute the exact values but approximate the real values of the matrix exponential. Because of the approximation techniques, even for small Markov times, most of the matrix is filled with very small numbers. Fig. 3.17A shows the matrix exponential of $E = (1.8 * (M - I))$ for Markov time=1.8. This may create many unnecessary links within the network. We choose to filter unnecessary values using the ceil value of Markov time, so the InfoMap optimization algorithm does not need to compute the value of codelength gain for every pair of nodes.

The main idea of the filter is that, the matrix exponential at time t=1.8, cannot be greater than the power of the matrix for the ceil value of t, which is in this case 2. So we get the power of matrix for the ceil value of t, and create a one-zero filter matrix, F, which indicates whether the cell has a value or not as shown in Fig. 3.17B. Then we filter the matrix exponential matrix E using the filter matrix F, by multiplying elements of two matrices such as $E[i,j] * F[i,j]$, and we get the filtered exponential matrix E' as shown in Fig. 3.17C. Because of the filter, the sum of each row $sum(E'[i])$ is less than 1. In order to make the matrix again row-stochastic, for each row i, we distribute $(1 - sum(E'[i]))$ to the cells of the row according to their weight $(E'[i,j]/sum(E'[i]))$, and we get the matrix E'' in Fig. 3.17D.

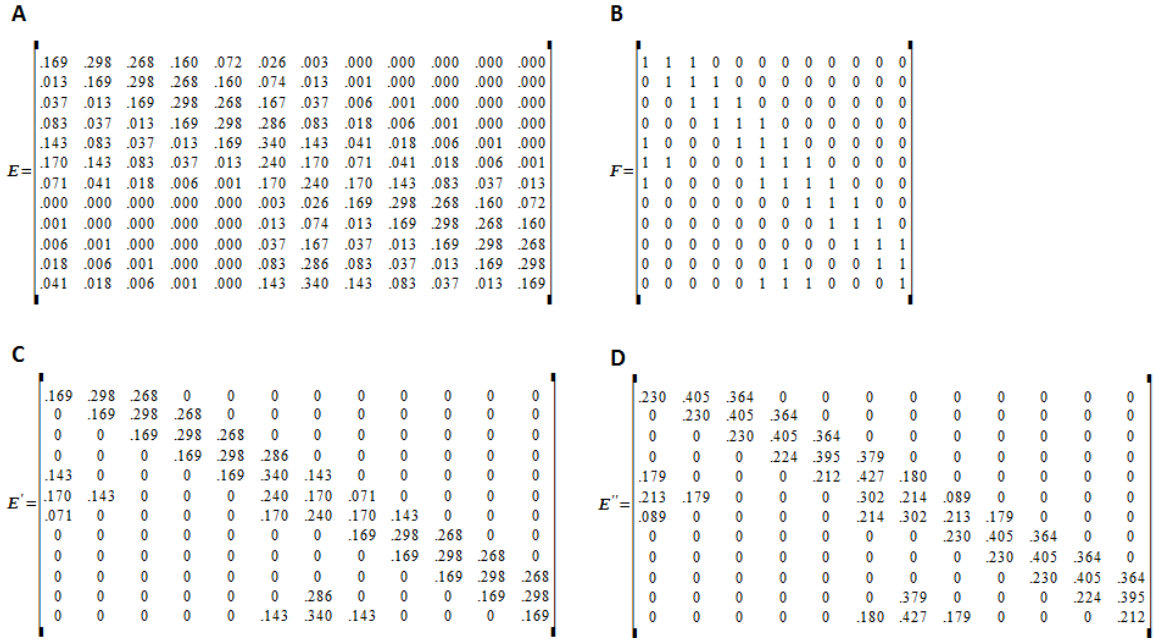


Figure 3.17: Filter matrix for the matrix exponential. (A) E=expm((M-I)*1.8), the matrix exponential of (M-I) at Markov time 1.8. (B) F, the filter matrix, (C) E' is the filtered E. (D) We added the remaining values to the E' and E'' becomes row-stochastic.

Now, we have the final transition matrix E'' for Markov time 1.8. Then we generate a new network using this transition matrix and apply the original InfoMap to find the communities of this network. Fig. 3.18A shows the transition matrix and the new network for Markov time 1.8. This process continues for all Markov times that we specify at the beginning, in this case for 100 time points between 0.1 and 100 in log space. As we can see from the result in Fig. 3.18B, InfoMap can find two communities after Markov time 1.

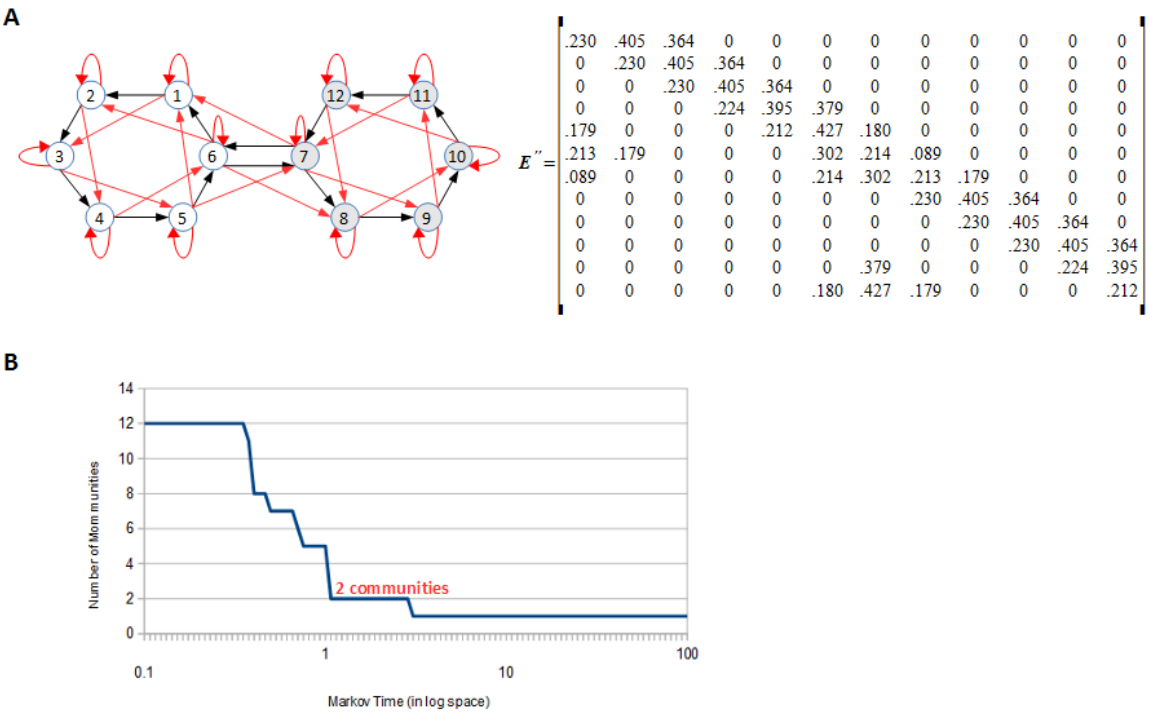


Figure 3.18: Time-dependent transition matrix. (A) A new network generated based on transition matrix at t=1.8. (B) Two communities can be found after Markov time 1.

One problem that we encountered in the analysis of real world directed networks is dangling nodes. In real world networks, some nodes may have no outgoing links, in this case the row of transition matrix for that node is all zeros. This makes the transition matrix not stochastic. Sometimes even adding a small noise may totally change the topology of the network when we get the exponential of the matrix, and this makes the matrix misleading [7]. It really depends on the domain knowledge and the nature of the data. We think, in our case adding a selflink to dangling nodes makes the transition matrix stochastic and also does not change the topology or nature of the matrix when we get the exponential of the matrix. We tested this approach on some networks by adding a selflink to dangling nodes and getting the matrix exponential of the matrix for different Markov times. We didn't see any misleading result or big difference between the topology of the original network and the one generated by the new transition matrix.

In terms of analysis of Markov Dynamics over time: at time=0, each node has its own community, and the number of communities equals the number of nodes. For discrete-time Markov Dynamics, time=1 equals the original InfoMap. For continuous-time Markov dynamics, because of the approximation techniques of matrix exponential, the results at time=1 may not be equal to the

original InfoMap, but the intermediate times finds what the original InfoMap finds and also other possible structures. Finally at time=∞, there will be one community if there are no unconnected components, otherwise the number of communities will be equal to the number of unconnected components or subgraphs.

As a result, Markov Dynamics is a diffusion process, and can be used as a zooming lens to detect the communities of a network at multiple scales [28]. It doesn't give you exact results, but can be used more like an exploratory tool that shows the communities at different scales. One can analyze the network for different time intervals, for example first between 0.1 and 100 for 100 time points, and if there are some interesting results in between then the analysis can be focused on that region, e.g. between 1 and 10 for 100 time points, and so on.

3.3 Overlapping Community Detection

Until now, we have studied the hard partitioning where each node can only be a member of one community. But in real life, a person may have different roles in many communities or a protein may have separate functions in different pathways. Therefore we need overlapping community detection algorithms that create soft partitions where a node can be a member of more than one community.

Two different approaches have been introduced to InfoMap for detecting overlapping communities: link clustering and compression of flow to reveal the overlapping communities [13, 9]. The first approach runs InfoMap to find the communities of links instead of nodes. It uses hard partitioning where each link can belong to only one community. Then, all the boundary nodes between link communities are assigned multiple membership for the neighboring communities. Because of its nature, it can be used for communities with intense overlap, as it will assign every boundary node to multiple communities in a coarse manner [13].

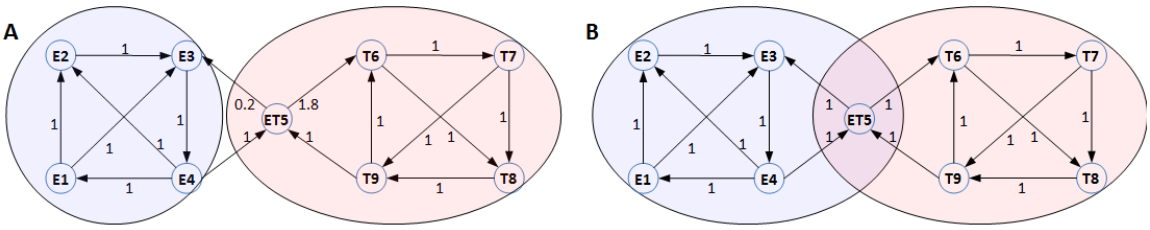


Figure 3.19: Proportion of returning flow affects the flow compression. (A) If the proportion of returning flow is low, flow compression can be better with hard clustering. (B) If the proportion of returning flow is high then we can better compress the flow with two overlapping communities.

The second approach uses a more fine tuning algorithm, and we choose to implement this approach with minor changes from the original one as explained in [9]. The key idea is to take advantage of regularities in the boundary flow between the communities. The pattern of the flow on the boundary node determines whether we should do hard partitioning or soft partitioning by assigning the node to both communities. If the proportion of the returning flow is low as seen in Fig. 3.19A, it means the boundary node is used as a transit point, and the best flow compression can be obtained by hard clustering, assigning the node to one of these communities. On the other hand, if the returning flow is high as in Fig. 3.19B, then the node may be part of both communities. In this case, we can obtain the best flow compression by soft clustering, assigning the node to both communities.

We will illustrate how overlapping communities can better compress the flow on a simple network with 9 nodes and 25 arcs as depicted in Fig. 3.20. Assume we have a network with English and French words and a common word used in both languages. A random walker visits the nodes of this network according to the outgoing probabilities and generates a text that is composed of English and French words consecutively as shown in Fig. 3.20A. How can we find the overlapping communities: English words, French words, and the common word by using the InfoMap's two-level compression? If we run the original InfoMap where a node can belong to only one community, it detects two communities with average codelength per step to 2.952 bits as shown in Fig. 3.20B.

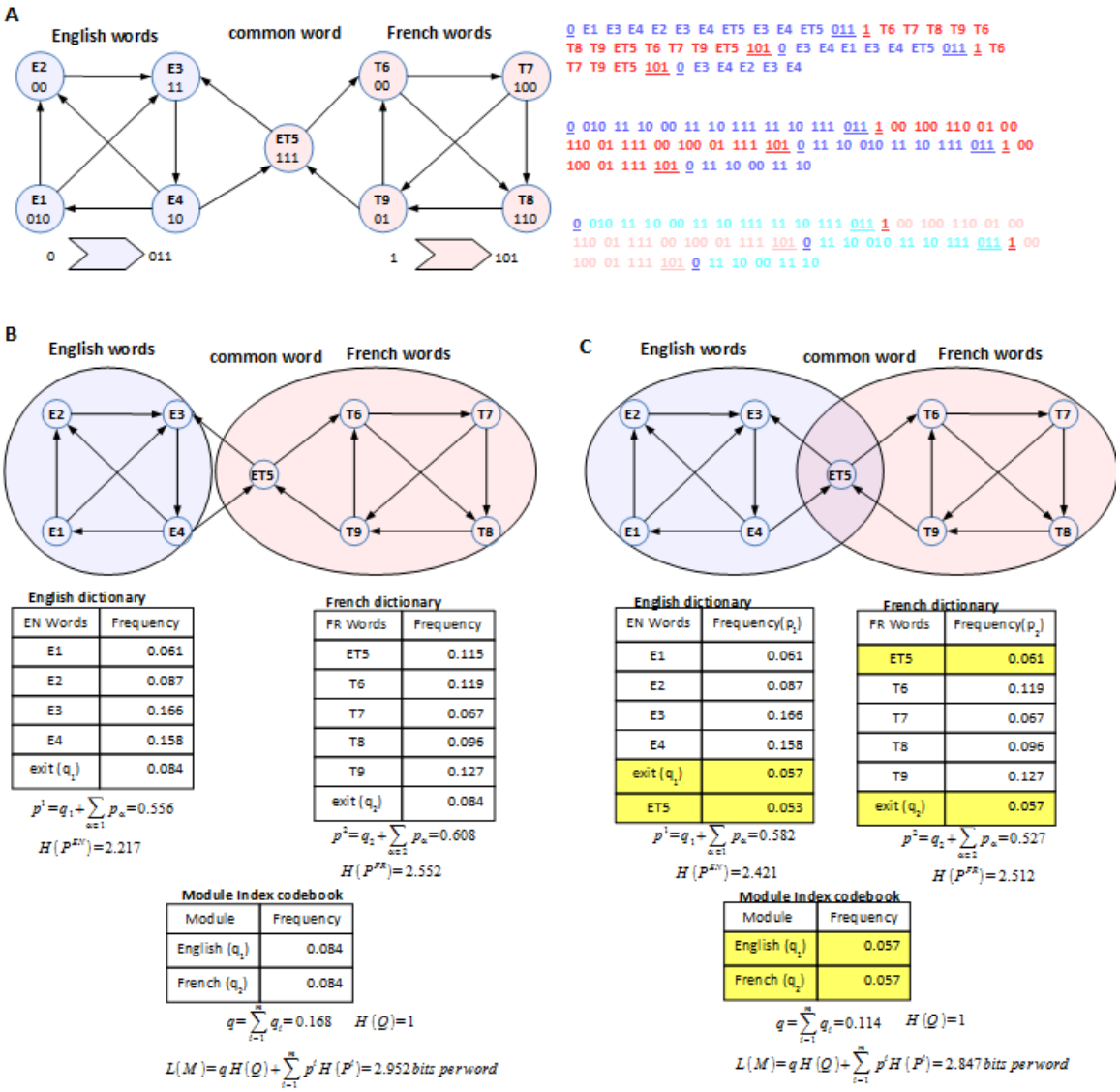


Figure 3.20: Detecting overlapping communities using the compression of flow. (A) We are trying to compress the trajectory of a random walker on the network with English words, French words and a common word. (B) The original InfoMap finds two non-overlapping communities with the average codeLength of 2.952 bits per word. (C) By exploiting the regularity of flow on the boundary flow, we can compress the trajectory better with two overlapping communities and decrease the average codeLength to 2.847 bits per word.

For this particular network, the returning flow is dominant on the boundary node ET5. If we exploit this regularity in the boundary flow and allow multiple membership to nodes then we can compress the flow better with overlapping communities. In Fig. 3.20C, ET5 is a member of

both the English dictionary and the French dictionary. Since the flow is ergodic, the ergodic node visit frequency of ET5 is fixed, and it should be shared among the overlapping modules with their fraction of incoming flow. Therefore (inFlowEnglish/inFlowAllModules) of its size belongs to the English module, and (inFlowFrench/inFlowAllModules) of its size belongs to the French module. ET5 is on the boundary and its size on both modules decreases. Because of that the exit frequency of both modules and also the frequency of modules in the index codebook decrease. With this overlapping assignment, the average codelength per step decreases to 2.847 bits.

To find the best partitioning for the overlapping communities, we use a similar optimization algorithm as explained in [9]. First the original InfoMap runs on the network and finds non-overlapping communities. We start with this partitioning, and apply the following procedure for all nodes. We choose a boundary node and measure the change in codelength by assigning it to one of its neighboring communities without removing from its own community, one by one for all neighbors. Then the node will have multiple membership with the neighboring communities that results in positive gain in codeLength. The boundary node may become a member of more than two communities. If there is no positive gain or it is less than a threshold value, the node stays in the same module. Since we start with the output of the original InfoMap, overlapping communities are heavily dependent on the initial partitioning, and may not find intense and deep overlaps between communities.

Chapter 4

Results

We applied the multiscale overlapping community detection algorithm to three different kinds of datasets. First we applied this to a set of synthetic networks to verify that the method works as a proof of concept. Then we tested the method with the benchmark graphs generated by a program to compare the results with other methods [18]. Lastly we used the method to analyze a socio-technical network called Tapped In to show that it can be applied to a real world network [33]. In this section, we will discuss the results we obtained from these datasets.

4.1 Analysis of Synthetic Networks

We constructed a set of synthetic networks to find the communities of a network at multiscales and with overlapping parts. In Fig. 4.1, we analyzed a directed network of 27 nodes and 39 arcs that has a two-level hierarchical structure. We ran the program for 100 time points between 0.1 and 100 in log space, and it detects the number of communities for each time point. The program does not give you an exact result or the number of communities found, but it can be used as an exploratory tool. When we generate a line chart using the number of communities at each time point as shown in Fig. 4.1, we can see the trend in the timeline. Nine communities are found between time points 0.57 and 1.87, and three communities are found between time points 2.84 and 8.69. Therefore we can conclude that this network has two different community structures with nine partitions and three partitions. The method can be used to detect multi-level community structures whether they are hierarchical or not.

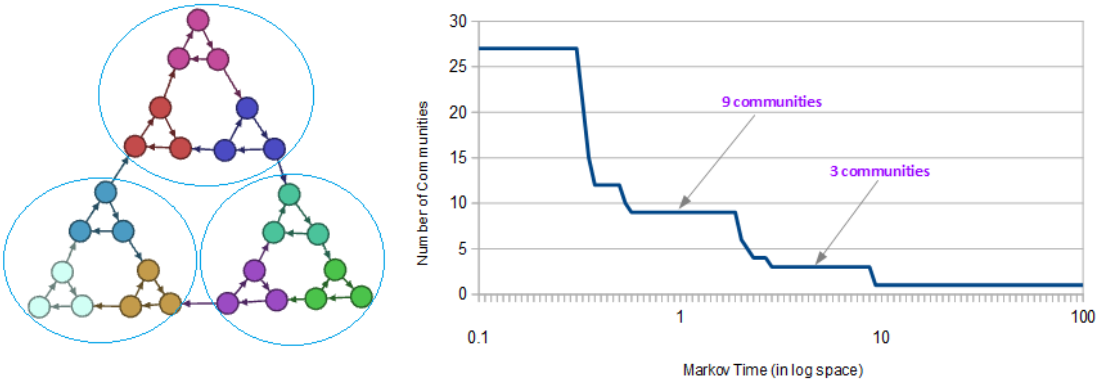


Figure 4.1: A directed network with 27 nodes and 39 arcs that has a two-level community structure. Nine communities and three communities can be found using the multiscale community detection method. Drawing is inspired from [26].

The usage of multiscale community detection is not only limited to finding hierarchical or multi-level communities. In some cases, the network may have a single community structure but it may not be detected using single-step methods such as the original InfoMap or others, because single-step methods cannot make use of the full connectivity pattern of the network or are biased to find clique-like communities. Therefore they cannot find communities with large diameter or unexpected topology such as rings or stars. In Fig. 4.2, we analyzed a directed network with 100 nodes and 105 arcs that has a ring like community structure. Again we ran the program for 100 time points between 0.1 and 100 in log space and generated a line chart using the number of communities found at each time point. Since Markov time 1 equals the original InfoMap, we couldn't find the five communities at Markov time 1. But when we analyze the trend in the timeline, we can see that five communities were found in a long range between the time points 3.05 and 16.29. Therefore we can conclude that the network has a community structure with five communities.

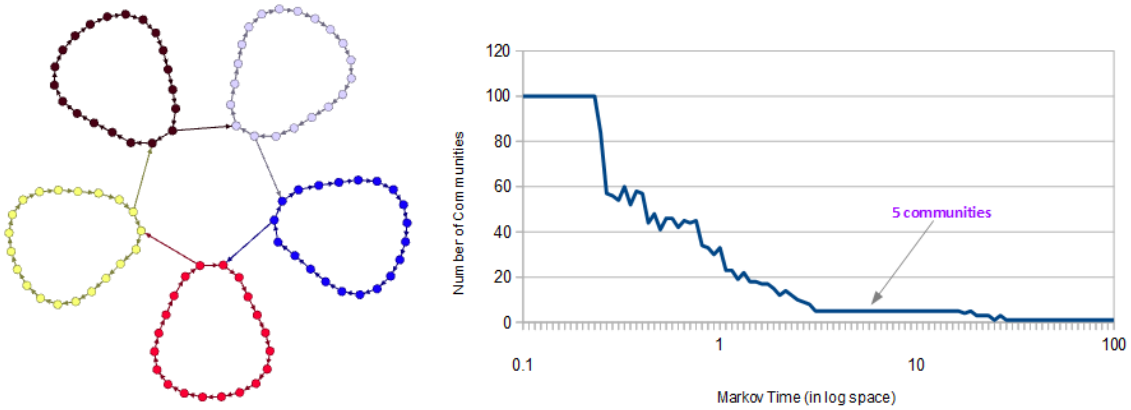


Figure 4.2: A ring-like directed network with 100 nodes and 105 arcs. Five communities can be found after Markov time 3 using the multiscale community detection method. Drawing is inspired from [28].

Next, we tested the program for overlapping community detection. In Fig. 4.3, we analyzed a directed network of 13 nodes and 24 arcs that may have overlapping communities.

First we ran the original InfoMap on this network, which has a constraint that a node cannot be a member of more than one community. It finds three separate communities by compressing the flow with the average codelength per step of 3.19963 bits as shown in Fig. 4.3A. Then we removed this constraint and allowed nodes to have multiple memberships as described in section 3.3. When we ran the program on the same network, we found three overlapping communities with better compression flow. The average codelength per step decreases to 2.68824 bits as in Fig. 4.3B. In some networks where the boundary nodes of communities have a high fraction of returning flow, we can better compress the flow by partitioning the network into overlapping communities.

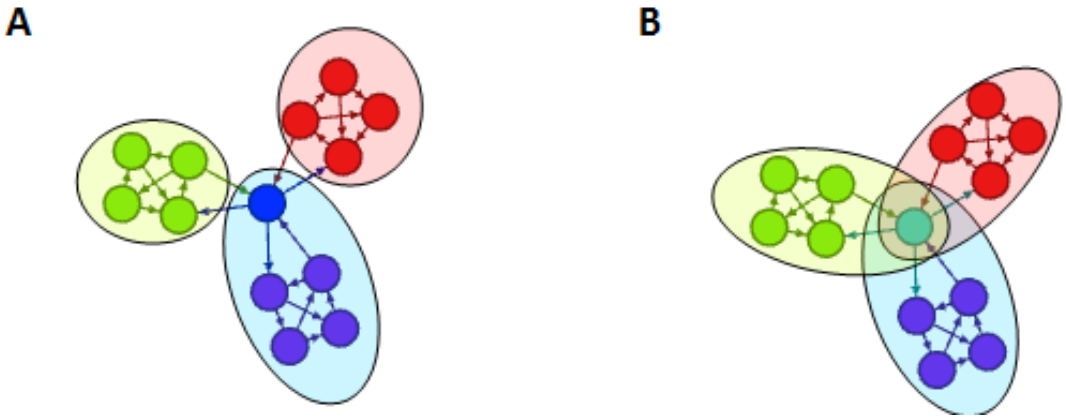


Figure 4.3: A directed network with 13 nodes and 24 arcs. (A)The average codelength per step is 3.19963 bits for this particular hard partitioning. (B)Partitioning into three overlapping communities decreases the average codelength per step to 2.68824 bits.

4.2 Analysis of Benchmark Networks

The performance of new community detection methods are usually measured with constructed synthetic datasets, or a special real world network where community size and degree size of nodes are close to homogenous. But many real life networks have heterogenous structure in terms of community size and degree size, which have a skewed degree distribution with a long tail. Therefore the LRF benchmark has been introduced to generate networks with heterogenous community and degree sizes according to a power law. It has parameters that allow you to set the exponential value for the power law distribution of community size and degree size, and also a mixing parameter that specifies the fraction of links inside the communities [18, 16, 17].

Comparing methods with each other is not easy and may not be accurate. Even the LRF benchmark may not be very accurate and cannot reflect all properties of real world networks, because it generates networks according to a set of specified parameters. But it may help us to find the limits of a method in terms of network size, community size, mixing parameter, degree sequence etc [17]. Therefore we choose the LRF benchmark graphs for benchmarking.

As described in [17], we used the normalized mutual information to measure the similarity between the planted partition and the recovered partition. It is used widely in the literature and also has extensions for overlapping communities and hierarchical communities. Normalized mutual information takes values between 1 and 0. If two partitions are identical, then the normalized mutual information between them is 1, or if they are totally different such as one partition has only

one community then the normalized mutual information between them is 0. Assume that we have two partitions X and Y. Then the normalized mutual information between them is defined as:

$$I_{norm}(X,Y) = \frac{2I(X,Y)}{H(X)+H(Y)} = \frac{2(H(X)-H(X|Y))}{H(X)+H(Y)} \tag{4.2.1}$$

where H(X) and H(Y) are the entropy of community sizes in partition X and Y respectively, and $H(X|Y)$ is the conditional entropy of X given Y. We show the computation of normalized mutual information for a simple case in Fig. 4.4.

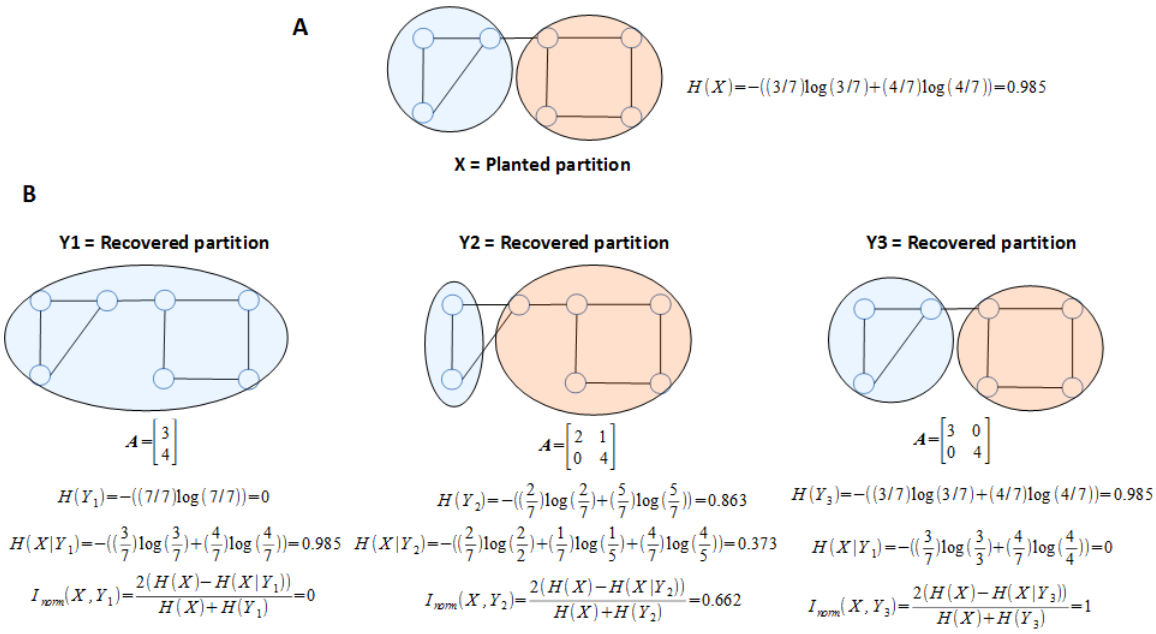


Figure 4.4: (A)The planted partition X, and the entropy of community sizes within this partition. (B)We find the similarity of planted partition to three other recovered partition using Normalized Mutual Information. The rows and the columns of this matrix shows the community size of planted partition and recovered partition respectively. The values of the matrix indicates the number of overlapping nodes in both partitions.

From now on, we will refer to the our extension to InfoMap as InfoMapMarkov. First we compare the performance of InfoMapMarkov with the Hierarchical InfoMap on directed unweighted networks. InfoMap uses two-level compression to find the communities of a network, whereas Hierarchical InfoMap has no constraint on the number of levels, thus it uses multilevel compression [26].

We used the extension of the LFR benchmark that generates hierarchical two-level networks: coarse-level and fine-level (https://sites.google.com/site/santofortunato/inthepress2). As ex-

pressed by its authors, the LFR benchmark is based on the balance between internal and external links, and might not seem to be a good benchmark for flow based methods such as InfoMap [16]. But flows can be generated using proper constraints, and Rosvall et al. developed these constraints in [26]. We used these constraints to generate well-defined two-level directed networks that are assumed to have a flow.

We generated directed unweighted networks with 1200 nodes, average degree 10, maximum degree 40, and the coarse-level module size between 100 and 400, and fine-level module size between 10 and 50. The LFR benchmark uses a power law distribution to generate scale free networks. We chose (exponent -2) for the degree size distribution and (exponent -1) for the module size distribution both in coarse-level and fine-level. For our experiments, we used the same coarse-level mixing parameter $\mu_1 = 0.1$, which is the fraction of links between nodes belonging to different macro-communities. We generate the line chart for varying values of the fine level mixing parameter μ_2 from 0.1 to 0.6, which indicates the fraction of links between nodes belonging to the same macro but not micro community. Therefore (1 - μ_1 - μ_2) is the fraction of links within the same micro and macro community.

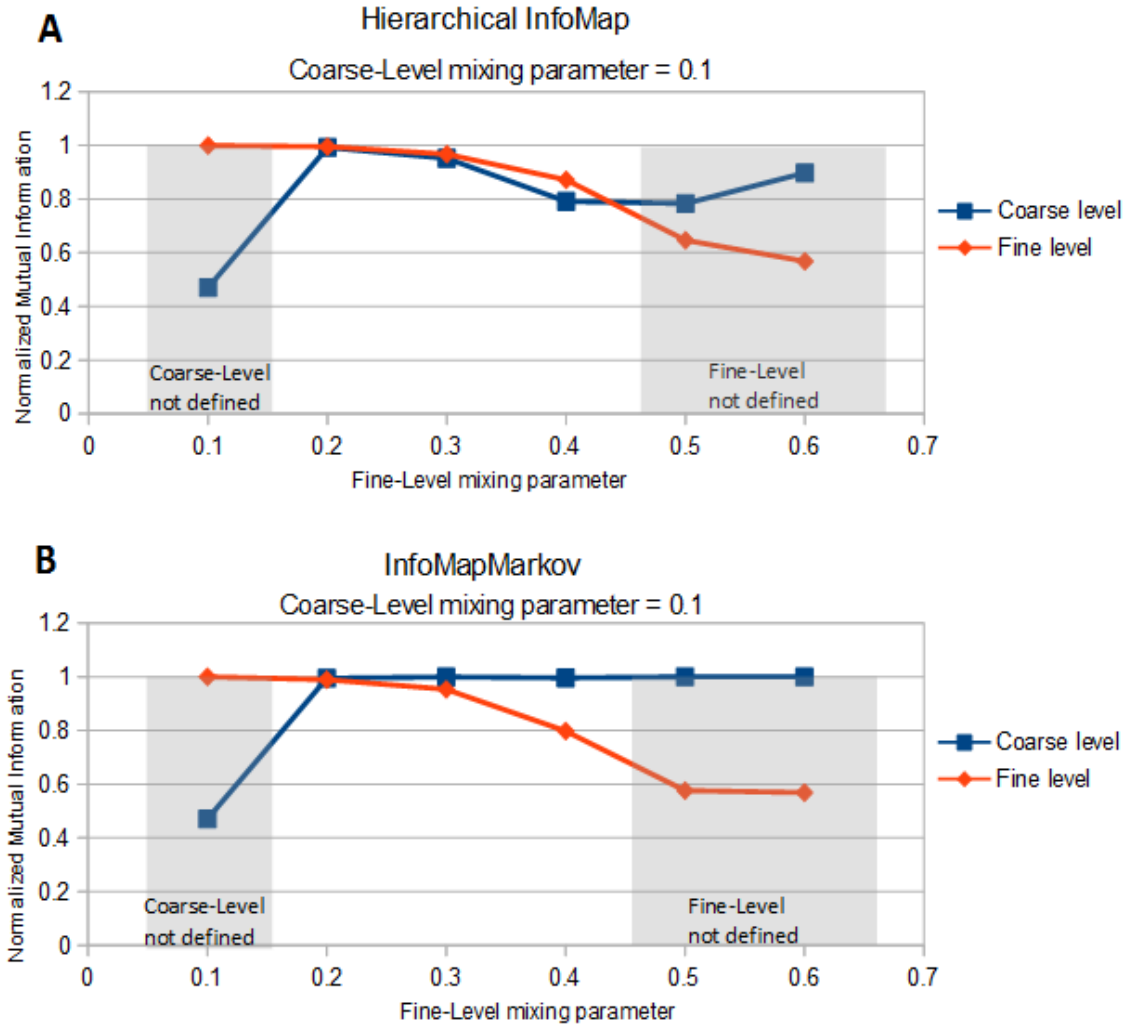


Figure 4.5: Benchmark test for multilevel community detection. Each point represents the average of 10 networks with the same parameters. The coarse-level mixing parameter, $\mu_1 = 0.1$ is fixed, fine-level mixing parameter μ_2 varying between 0.1 and 0.6. The performance of (A) Hierarchical InfoMap and (B) InfoMapMarkov for a network of 1200 nodes.

We compare the partitions found by InfoMapMarkov and hierarchical InfoMap with the benchmark planted partition separately as shown in Fig. 4.5A-B. Also we compare the performance of the communities found at the coarse-level and the fine-level independently. Each point is generated by taking the average of 10 different networks with the same parameters. In Fig. 4.5, the closer the Normalized Mutual Information is to 1, the better the recovery of the planted communities. Values are good only for mid-range fine-level mixing.

Both methods (InfoMapMarkov and hierarchical InfoMap) could not find the coarse-level communities at $\mu_2 = 0.1$; they found only one-level partitioning. Because the fraction of links between macro communities and micro communities are the same $\mu_2 = \mu_2 = 0.1$, the methods cannot distinguish macro communities. Similarly both methods could not find the fine-level communities at $\mu_2 = 0.5$ and $\mu_2 = 0.6$, where they only found one-level partitioning. Because after $\mu_2 = 0.4$, the fraction of links within the fine-level modules (1 - μ_1 - μ_2) is less than the fraction of links between the fine-level modules, consequently the organizational structure of fine-level communities becomes very weak.

Between $\mu_2 = 0.2$ and $\mu_2 = 0.4$, both methods detected two-level partitioning, because the hierarchical organization is well defined for these values. Well defined organization means: the fraction of links within micro communities is greater than the fraction of links between micro communities, and that is greater than the fraction of links between macro communities [26]. When the methods found only one partition at $\mu_2 = 0.1$, $\mu_2 = 0.5$ and $\mu_2 = 0.6$, we compare the found partition to both planted partitions (coarse-level and fine-level) to generate the line chart in Fig. 4.5.

Overall, the performance of InfoMapMarkov and hierarchical InfoMap is similar. But hierarchical InfoMap is slightly better at finding fine-level communities and InfoMapMarkov is slightly better at finding coarse-level communities. InfoMapMarkov runs the original InfoMap for 100 different Markov times, and we decide that a real partition is found if there is a trend in the timeline: the same partition appears several times. Also Barahona et al. introduced a metric called compression gap, which indicates a significant partition [28]. Compression gap measures the difference between the compression achieved by InfoMap and the theoretical limit given by the true entropy of the Markov process. Compression gap works in very well-defined synthetic networks, but we couldn't get good results on our benchmarks. Sometimes the same partition appears very few times; therefore, we may not choose the best Markov time point for the fine-level partition. This may explain the lower performance of InfoMapMarkov at the fine-level, and also the need for a metric to indicate the best partitions among all Markov time points.

Now we will compare the runtime performance of both methods. The complexity of InfoMap or hierarchical InfoMap increases linearly with the size of the network, and runs very fast even for large networks. Since we introduce Markov dynamics to InfoMap, we are getting the matrix exponential of the transition matrix at each time point. Thus it requires matrix multiplication, and the complexity of InfoMapMarkov increases quadratically or cubically according to the chosen implementation of matrix multiplication. As a practical example, we run the InfoMapMarkov on a real network called Tapped In with $\approx$ 12,000 nodes on a XServe machine with the following speci-

fications:

- Model Name: Xserve
- Model Identifier: Xserve1,1
- Processor Name: Dual-Core Intel Xeon
- Processor Speed: 3 GHz
- Number Of Processors: 2
- Total Number Of Cores: 4
- L2 Cache (per processor): 4 MB
- Memory: 16 GB
- Bus Speed: 1.33 GHz
- Memory speed: 667 MHz

The algorithm [19] we are using for matrix exponential takes 15 matrix multiplications, and each matrix multiplication takes $\approx$ 3 minutes on this server. Therefore the runtime for one Markov time took $\approx$ 45 minutes, and we ran it for 100 Markov times, so the overall runtime took $\approx$ 75 hours. The InfoMap optimization algorithm is fast, and its runtime is negligible. The matrix exponential makes the program very slow for large datasets, and we need faster approximation algorithms for matrix exponential, or a greedy approach to approximate the matrix exponential of higher values using the lower values without recomputing from the beginning.

Our second benchmark is for detecting overlapping communities. Again we used the LFR benchmark to test our overlapping community detection feature. In this case, we only tested the performance of the overlapping feature without the multiscale feature. We generated directed unweighted networks with 1000 nodes, average degree 10, maximum degree 50 and module size between 20 and 100. We choose the number of memberships of the overlapping nodes = 2, mixing parameter $\mu = 0.1$, exponent -2 for the degree size distribution and exponent -1 for the module size distribution.

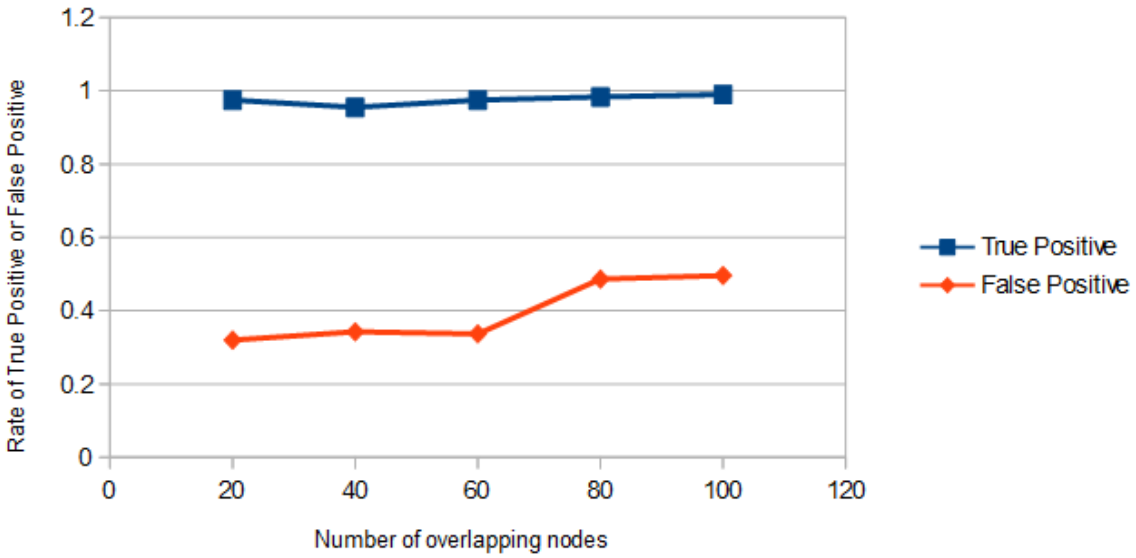


Figure 4.6: Benchmark for overlapping community detection. The coarse-level mixing parameter, $\mu_1 = 0.1$ is fixed, fine-level mixing parameter μ_2 varying between 0.1 and 0.6.

Fig. 4.6 shows the result of the benchmark. We measure the True Positive and False Positive rates between the number of overlapping nodes planted into the network and found by our extension to InfoMap. These measures are calculated as below:
True positive = # of true overlapping nodes found / # of overlapping nodes planted
False positive = # of false overlapping nodes found / # of overlapping nodes planted

As the number of overlapping nodes increase, the true positive rate remains the same but the false positive rate increases. We can find the planted overlapping nodes, but also many other nodes are counted as overlapping, falsely. We think that, finding overlapping communities by flow compression is useful for networks that have sparse overlappings. But, when the network has dense overlapping structure, the link community approach may give better results [13].

4.3 Preliminary Analysis of Tapped In Chats Network

Tapped In (www.tappedin.org) was a socio-technical network for education professionals that provided professional development, peer support, and collaborative study between them. Tapped In hosted content and activities of more than 150,000 education professionals and more than

50 “tenant” organizations such as education agencies and higher education institutions, and also volunteer participants. These organizations used Tapped In as an environment to support faculty and students for online courses, workshops, seminars and other collaborative activities. Volunteer members designed 40-60 community-wide activities per month to foster interaction between members. Most of the activities in Tapped In were also carried out by volunteers. Members could interact with each other using the features such as text-chat, discussion forums, sharing files with each other, and other tools [32, 5, 31, 33].

The Tapped In network generated in [33] is a “bipartite multimodal directed weighted graph”, also shortly called an associogram. All vertices of this graph are either an actor or an artifact. The graph is bipartite, edges are between actors (people) and artifacts. It is multimodal as there are three different kinds of artifacts: chat rooms, discussion forums and files. It is directed: arcs go from an actor to the artifact if the actor reads that artifact, and from the artifact to the actor if the actor modifies (creates/edits) that artifact. Lastly, the weights indicate the number of events that happened between the actor and the artifact. The graph is generated by the interaction of Tapped In members between September 2005 and May 2007. It has 35,012 vertices and 179,703 edges. The vertices contains 17,619 actors, 9896 discussion forums, 4845 files, and 2652 chat rooms [33].

We analyzed a subset of Tapped In network, which is called TI chats and includes only actors and chat rooms. This network has 12,833 nodes and 38,933 arcs. First, we ran the classic InfoMap, and it found 972 communities. Then we ran hierarchical InfoMap, and it found a multilevel community structure: 1085 communities at the fine-level, and 27 large communities at the coarse-level whose size is greater than 1 percent of total number of nodes. Finally, we applied our multiscale feature by running the algorithm for 100 time points between 0.1 and 100 in logspace. Fig. 4.7 shows the number of communities found at each point. As this is a real-world network, and of course the community structure is not well defined, we couldn’t see a multilevel community structure at first. Around 1000 communities were found around time 5, but there may be a multilevel community structure too. In order to reveal the multi-level community structure, we may need to analyse the number of communities whose size are greater than a treshold, as hierarchical InfoMap does.

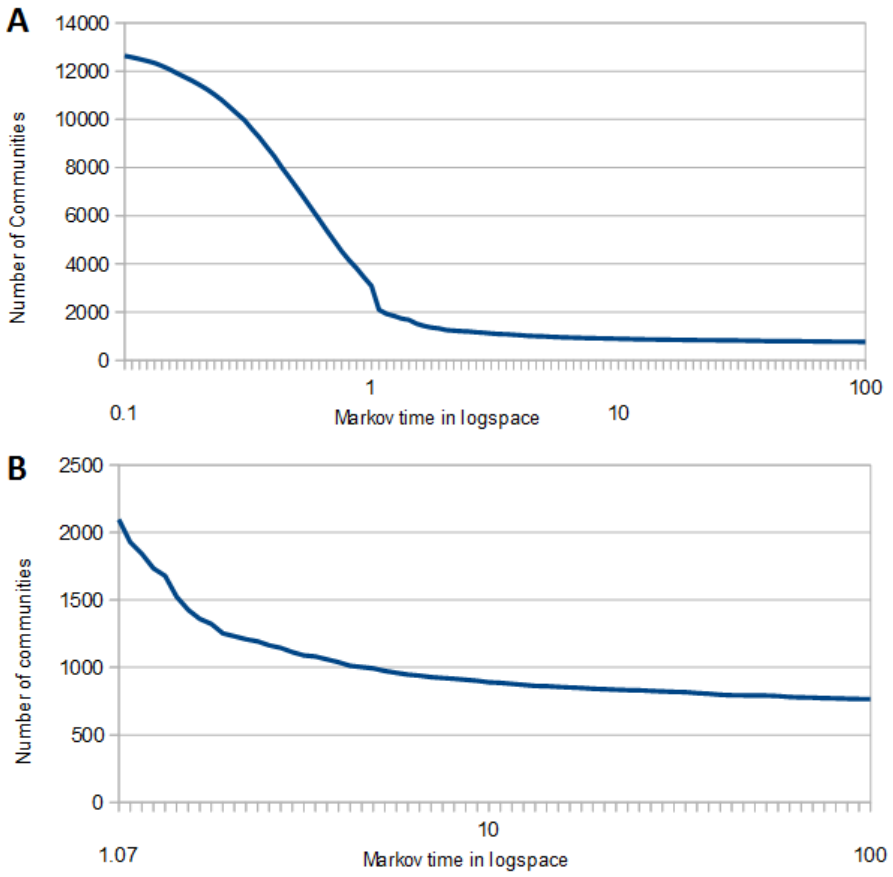


Figure 4.7: Community analysis of TI-chats at multiscales: (A) The number of communities between 0.1 and 100. (B) Zoom in the number of communities between 1 and 100.

Secondly, we ran our overlapping feature for TI-chats network, starting with the hard partitioning of InfoMap. We found 30 nodes that have multiple memberships as shown in Fig. 4.8, and they are sorted by the gain in codelength from the highest to the lowest.

node id	node name
93	“47”
85	“8”
30	“ROOM 97”
264	“88876”
271	“29595”
97	“1082”
948	“ROOM 1350”
49	“ROOM 15”
463	“33399”
48	“ROOM 95”
51	“ROOM 12688”
1152	“28521”
2546	“22913”
39	“ROOM 32”
905	“ROOM 14665”
268	“61194”
1396	“ROOM 988”
260	“ROOM 4578”
2737	“22301”
454	“ROOM 3674”
58	“ROOM 3275”
449	“11162”
812	“25383”
1241	“23”
2488	“ROOM 15266”
190	“34830”
558	“1236”
77	“529”
2478	“ROOM 858”
389	“549”

Figure 4.8: Overlapping community analysis for TI-chats: We found 30 nodes that have multiple memberships.

JeffC (node 47) and BjB (node 8) were the two most active users in the system, and were volunteer facilitators. (These users have given us permission to identify them; others will remain anonymous.) BjB was so active that she was given a second user account, BJB2 (node 29595), that we can see later on the list, so she could facilitate in two rooms at once. She was the only user who had that ability. These actors worked in the Tapped In reception room, sent people to the correct room for events, and helped event leaders run events across a variety of topics (hence multiple communities). The fact that they are ranked the highest shows we can find the most important bridges.

The most active room was actually “ROOM 1”, The Tapped In Reception room. However, it was removed from the TI-chats network as almost everyone passed through it so it would dominate the community detection. “ROOM 97” is the after school online room, where many scheduled chat events took place. This room was used by many communities in the Tapped In network, and

is by far the most active room in the system, so is expected to be categorized in multiple graph communities. The high ranking shows the algorithm finds an important bridge. “ROOM 1350” is the office of a faculty of teacher education at a tenant university, who probably ran classes here with a diverse audience. “ROOM 15” is BjB’s office, and she had many diverse conversations in this office. “ROOM 95” is the lobby of the Tapped In Groups Floor. Many people would have gone through this room on their way to different group rooms, so this room would be connected to the various communities associated with those rooms. “ROOM 32” is JeffC’s office, which he used for “Sustained Online Support”. Thus, persons associated with many other communities would have come to this room for support. There were similar explanations for the multiple community membership of other actors and rooms that we examined in this list. The overlapping community detection feature exposes important actors that community managers would want to know about.

Chapter 5

Conclusion and Future Work

In this thesis, we introduced two features to the InfoMap community detection method. First, we introduced Markov Dynamics to InfoMap in its original C++ language to detect communities at multiscales in directed networks. Previous work introduced Markov Dynamics to InfoMap in Matlab for undirected networks [28]. Secondly we added the feature to find overlapping communities by compressing the flow, following the approach explained in [9].

Based on our benchmark results, InfoMapMarkov can be used as an alternative to Hierarchical InfoMap. Its performance for detecting coarse-level communities is especially promising. Also it can be used more as an exploratory tool or zooming lens to analyze directed networks at different scales. However, InfoMapMarkov needs improvement in two areas:

Runtime: Taking the matrix exponential of large matrices is time-consuming and computationally costly. We need a faster approximation algorithm for the matrix exponential, or a greedy approach. Computing the matrix exponential for a few Markov times, and generating the other ones by using the already computed ones may be an alternative approach.

Indicator metric: For synthetic networks or very well-defined benchmark graphs, we can easily detect the significant communities found by analysing the trend among the timeline. Also the compression gap metric as explained in section 4.2 works for them. But real world networks or some benchmark graphs do not have very well defined organizational structures. Because of that, we often may not see a trend in the timeline such as the same partition found several times consecutively. Therefore we need a metric that indicates the Markov times that have a significant partition.

In future work, we are planning to test on more types of benchmark graphs and real world networks with larger size. So we can better evaluate the performance of our extension to InfoMap.

Additionally, we want to compare the results with other algorithms to evaluate the limits of the program.

In conclusion, community detection is one of the most commonly used technique to analyze social, biological or technological networks. It allows experts to have more insight about the network data. Besides this, sequence analysis is also an important problem in data analysis. Genomic sequences or text sequences have many similarities with the large scale network data. Many genomic data sequences or texts have hierarchical structure or have various structures/functions at different scales. We think that sequences can be represented as a graph and community detection algorithms can be applied to find structure or functions of sequence data at multiscales.

Bibliography

[1] Y.Y. Ahn, J.P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. Nature, 466(7307):761–764, 2010.

[2] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10):P10008, 2008.

[3] P. Boldi, M. Santini, and S. Vigna. Pagerank as a function of the damping factor. In Proceedings of the 14th international conference on World Wide Web, pages 557–566. ACM, 2005.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems, 30(1):107–117, 1998.

[5] K.H. Chu, D.D. Suthers, and D. Rosen. Uncovering multi-mediated associations in socio-technical networks. In System Science (HICSS), 2012 45th Hawaii International Conference on, pages 3560–3569. IEEE, 2012.

[6] A. Clauset, M.E.J. Newman, and C. Moore. Finding community structure in very large networks. Physical review E, 70(6):066111, 2004.

[7] J. Cook. Making a singular matrix non-singular. Website. `http://www.johndcook.com/blog/2012/06/13/matrix-condition-number`.

[8] J.C. Delvenne, S.N. Yaliraki, and M. Barahona. Stability of graph communities across time scales. Proceedings of the National Academy of Sciences, 107(29):12755–12760, 2010.

[9] A.V. Esquivel and M. Rosvall. Compression of flow can reveal overlapping-module organization in networks. Physical Review X, 1(2):021025, 2011.

[10] S. Fortunato. Community detection in graphs. Physics Reports, 486(3):75–174, 2010.

[11] S. Fortunato and M. Barthelemy. Resolution limit in community detection. Proceedings of the National Academy of Sciences, 104(1):36–41, 2007.

[12] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. Proceedings of the National Academy of Sciences, 99(12):7821–7826, 2002.

[13] Y. Kim and H. Jeong. Map equation for link community. arXiv preprint arXiv:1105.0257, 2011.

[14] R. Lambiotte, J.C. Delvenne, and M. Barahona. Laplacian dynamics and multiscale modular structure in networks. arXiv preprint arXiv:0812.1770, 2008.

[15] R. Lambiotte and M. Rosvall. Ranking and clustering of nodes in networks with smart teleportation. Physical Review E, 85(5):056107, 2012.

[16] A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Physical Review E, 80(1):016118, 2009.

[17] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. Physical review E, 80(5):056117, 2009.

[18] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. Physical Review E, 78(4):046110, 2008.

[19] P. Liu. Matrix exponential code. Website. `https://github.com/poliu2s/MKL/blob/master/matrix_exponential.cpp`.

[20] M.E.J. Newman. Modularity and community structure in networks. Proceedings of the National Academy of Sciences, 103(23):8577–8582, 2006.

[21] M.E.J. Newman. Networks: an introduction. Oxford University Press, Inc., 2010.

[22] M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical review E, 69(2):026113, 2004.

[23] M. Rosvall, D. Axelsson, and C.T. Bergstrom. The map equation. The European Physical Journal-Special Topics, 178(1):13–23, 2009.

[24] M. Rosvall and C.T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. Proceedings of the National Academy of Sciences, 104(18):7327–7331, 2007.

[25] M. Rosvall and C.T. Bergstrom. Maps of random walks on complex networks reveal community structure. Proceedings of the National Academy of Sciences, 105(4):1118–1123, 2008.

[26] M. Rosvall and C.T. Bergstrom. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. PloS one, 6(4):e18209, 2011.

[27] M.T. Schaub, J.C. Delvenne, S.N. Yaliraki, and M. Barahona. Markov dynamics as a zooming lens for multiscale community detection: non clique-like communities and the field-of-view limit. PLOS ONE, 7(2):e32210, 2012.

[28] M.T. Schaub, R. Lambiotte, and M. Barahona. Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation. Physical Review E, 86(2):026112, 2012.

[29] C.E. Shannon and W. Weaver. The mathematical theory of information. University of Illinois Press, 1949.

[30] H.A. Simon. The architecture of complexity. Proceedings of the American Philosophical Society, 106(6):467–482, 1962.

[31] D.D. Suthers and K.H. Chu. Multi-mediated community structure in a socio-technical network. In Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, pages 43–53. ACM, 2012.

[32] D.D. Suthers and C. Desiato. Exposing chat features through analysis of uptake between contributions. In System Science (HICSS), 2012 45th Hawaii International Conference on, pages 3368–3377. IEEE, 2012.

[33] D.D. Suthers, J. Fusco, P. Schank, K.H. Chu, and M. Schlager. Discovery of community structures in a heterogeneous professional online network. In System Sciences (HICSS), 2013 46th Hawaii International Conference on, pages 3262–3271. IEEE, 2013.

[34] R. Tanase and R. Radu. Pagerank algorithm - the mathematics of google search. Website. `http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html`.