

HGRN Algorithm

Jarred M. Kvamme¹, Boyu Zhang², and Audrey Q. Fu^{1,3}

¹*Department of Bioinformatics and Computational Biology -
University of Idaho*

²*Department of Computer Science - University of Idaho*

³*Department of Mathematics and Statistical Science - University of
Idaho*

October 13, 2023

TABLE OF CONTENTS

LIST OF TABLES	1
----------------	---

LIST OF FIGURES	1
-----------------	---

1 Preprocessing	2
-----------------	---

2 Training	2
------------	---

LIST OF TABLES

1 Notation and explanations	11
---------------------------------------	----

LIST OF FIGURES

1 Proposed HRGN model. Example represents a model constructed for a hierarchy with $\ell = 2$ super layers.	10
--	----

1 Preprocessing

1.1 Graph Initialization: Given an attributed network $\mathcal{G}_0(E, V, \mathbf{X})$ with attribute matrix $\mathbf{X} = \{x_1, x_2; \dots; x_N\} \in \mathbb{R}^{N \times p}$ where \mathbf{x}_i is the p -length attribute vector of node n_i , $E = \{(v_i, v_j) | 1 < i, j < N, i \neq j\}$ edges, and $V = \{v_i\}_{i=1}^N$ nodes/vertices, estimate an initial graph of \mathbf{X} represented by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$

1.1.1 Correlations method: Compute the correlation matrix $\mathbf{R} \in [0, 1]^{N \times N}$ from \mathbf{X} . Convert the correlation matrix \mathbf{R} into an adjacency matrix \mathbf{A}_0 such that

$$\hat{a}_{i,j} = \begin{cases} 1 & \text{if } r_{i,j} > \rho \\ 0 & \text{else} \end{cases}$$

where ρ represents a minimum correlation threshold to consider an edge (v_i, v_j) between nodes i and j .

1.1.2 K-neighbors method:

1.1.3 Precision method:

1.2 Estimate Hierarchical Structure:

2 Training

2.1 Initialize HGRN Model

Input: The model takes as input the attributed graph represented by the adjacency matrix and node attribute matrix $\{\mathbf{A}, \mathbf{X}\}$, respectively.

Parameters: **Number of hierarchical layers** (ℓ), **communities per layer** ($K = \{k_i\}_{i=1}^\ell$), **max training epochs** (t)

Output: The output includes the reconstructed node attribute matrix $\hat{\mathbf{X}}$ and adjacency matrix $\hat{\mathbf{A}}$ as well as the node assignments the hierarchy **hierarchical graph** ($\mathcal{H} = \{\mathcal{G}_i\}_{i=0}^\ell$)

2.2 For t Epochs:

2.2.1 Compute forward pass:

Encoder Module:

For m encoder layers:

We adopt the graph attention autoencoder **GATE** proposed by [1] which consists of graph attention based encoder and decoder models that reconstruct the original graph \mathbf{A} and the node feature matrix \mathbf{X}

• Attention weights

The GAT mechanism applies additional trainable parameters \mathbf{a} to the neighbors of each node to learn their relevance given by the weight $\theta_{i,j}$. The attention weight $\theta_{i,j}$ represents the importance of node i in the

representation of j For the GAT model, the relevance of node neighbors (i.e their attention weights) is computed via the following:

$$\theta_{ij} = \frac{\sigma(\exp(\mathbf{a}_s^T \phi[\mathbf{W}h_i] + \mathbf{a}_v^T \phi[\mathbf{W}h_j]))}{\sum_{k \in \mathcal{N}(i)} \sigma(\exp(\mathbf{a}_s^T \phi[\mathbf{W}h_i] + \mathbf{a}_v^T \phi[\mathbf{W}h_k]))}$$

The weight parameters \mathbf{a}_s^T aim to capture additional semantic information in node i while the weight parameter \mathbf{a}_v^T captures semantic information in node j . Furthermore, the Softmax function is used to normalize the attention coefficients so that the coefficients in the neighborhood of node i sum to 1.

In linear form, the [matrix of] attention coefficients for the m layer of the model $\Theta^{(m)}$ are computed as

$$\begin{aligned}\Theta^{(m)} &= \text{Softmax}(\sigma(\mathbf{M}_s^m + \mathbf{M}_v^m)) \\ \mathbf{M}_s^m &= \mathbf{A} \odot [\mathbf{a}_s^{(m)T} \cdot \phi(\mathbf{W}_m \mathbf{H}_{m-1})] \\ \mathbf{M}_v^m &= \mathbf{A} \odot [\mathbf{a}_v^{(m)T} \cdot \phi(\mathbf{W}_m \mathbf{H}_{m-1})]^T\end{aligned}$$

where

$$\Theta_{ij}^{(m)} = \begin{cases} \theta_{ij}^{(m)} & \text{if there is an edge between node } i \text{ and node } j \\ 0 & \text{else} \end{cases}$$

Note that in the above equations σ is the sigmoid logistic function, \odot denotes the element-wise product operation between matrices, and ϕ denotes the layer activation function. In the original formulation of GAT by [2], ϕ was the identity function, and LeakyReLU was used in place of the sigmoid function.

• **Compute encoder hidden layers** By considering $H_0 = X$, the $m - 1^{th}$ encoder layer generates node representations in hidden layer $m - 1$ as follows:

$$\mathbf{H}_{m-1} = \sigma(\mathbf{W}_{m-1} \cdot \mathbf{H}_{m-2}) \cdot \Theta^{(m-1)}$$

Where Θ_m is the matrix of attention coefficients for layer $m - 1$

- **Compute embedding dimension**

The embedding or bottleneck dimension follows the same format above as fully connected GAT layer which takes the (m^{th}) hidden layer of the encoder as input.

$$\mathbf{Z} = \sigma(\mathbf{W}_m \cdot \mathbf{H}_{m-1}) \cdot \Theta^{(m)}$$

Decoder Module:

For m decoder layers:

- **Attention weights**

The attention weight of the decoder layers are computed in similar fashion where $\theta_{i,j}$ represents the importance of node i in the representation of j :

$$\hat{\theta}_{ij} = \frac{\sigma\left(\exp\left(\mathbf{a}_s^T \phi\left[\hat{\mathbf{W}} \hat{h}_i\right] + \mathbf{a}_v^T \phi\left[\hat{\mathbf{W}} \hat{h}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}(i)} \sigma\left(\exp\left(\mathbf{a}_s^T \phi\left[\hat{\mathbf{W}} \hat{h}_i\right] + \mathbf{a}_v^T \phi\left[\hat{\mathbf{W}} \hat{h}_k\right]\right)\right)}$$

Where the matrix of attention coefficients for the m^{th} decoder layer is defined as:

$$\begin{aligned}\hat{\Theta}^{(m)} &= \text{Softmax}\left(\sigma\left(\hat{\mathbf{M}}_s^m + \hat{\mathbf{M}}_v^m\right)\right) \\ \hat{\mathbf{M}}_s^m &= \mathbf{A} \odot \left[\hat{\mathbf{a}}_s^{(m)T} \cdot \phi\left(\hat{\mathbf{W}}_m \hat{\mathbf{H}}_{m-1}\right)\right] \\ \hat{\mathbf{M}}_v^m &= \mathbf{A} \odot \left[\hat{\mathbf{a}}_v^{(m)T} \cdot \phi\left(\hat{\mathbf{W}}_m \hat{\mathbf{H}}_{m-1}\right)\right]^T\end{aligned}$$

where

$$\hat{\Theta}_{ij}^{(m)} = \begin{cases} \hat{\theta}_{ij}^{(m)} & \text{if there is an edge between node } i \text{ and node } j \\ 0 & \text{else} \end{cases}$$

- **Compute decoder hidden layers.**

The decoder layers also consist of m GAT layers under the GATE model architecture. We use $\hat{\mathbf{H}}$ notation to denote the layers of the decoder as reconstructions of the embeddings \mathbf{Z} . The first decoder layer takes the embedding matrix \mathbf{Z} as the feature information for the nodes and outputs the hidden decoder layer $\hat{\mathbf{H}}_1$

$$\hat{\mathbf{H}}_1 = \sigma\left(\hat{\mathbf{W}}_1 \cdot \mathbf{Z}\right) \cdot \hat{\Theta}^{(1)}$$

• **Reconstruct the input from the final decoder** following [1] we take the reconstructed node attributes as the final layer of the decoder:

$$\hat{\mathbf{X}} = \hat{\mathbf{H}}_m = \sigma\left(\hat{\mathbf{W}}_m \cdot \hat{\mathbf{H}}_{m-1}\right) \cdot \hat{\Theta}^{(m)}$$

- **Graph reconstruction** Following [3, 1, 4], we reconstruct the adjacency matrix of the attributed graph from the embedding dimension using a simple dot-product decoder function activation:

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z} \cdot \mathbf{Z}^T)$$

As usual, σ denotes the sigmoid (logistic) activation function which assumes a normal distribution and transforms the weights of the adjacency matrix into pseudo-probabilities of node linkages: $\hat{\mathbf{A}} \in [0, 1]^{N \times N}$

Super-Node Classification Layers:

For $\ell + 1$ hierarchical layers:

- **Compute community assignment probabilities**

In this step we construct ℓ single layer linear classifiers. Each linear classifier will output the community assignments of nodes (or super-nodes) from the previous layer. The first linear classifier uses the embeddings to classify N nodes to k_1 communities given by

$$\mathbf{P}_1 = \text{Softmax}(\mathbf{Z}\mathbf{W}_1)$$

$$= \begin{bmatrix} p_{1,1} & \cdots & p_{1,k_\ell} \\ \vdots & \ddots & \vdots \\ p_{k_{\ell-1},1} & \cdots & p_{k_{\ell-1},k_\ell} \end{bmatrix}$$

Where \mathbf{P} is a matrix with nodes in the rows and assignment probabilities in the columns. For example, the i row of \mathbf{P}_1 represent the probabilities for assigning node i to each of the k_1 communities. The Softmax function therefore regularizes the rows of \mathbf{P}_1 such that the sum of the probabilities equals one and each row represents a valid probability distribution for assigning nodes to communities.

- **Compute community assignment labels**

The community assignment matrix is a boolean matrix which represents the community assignment of a node or super node such that the $c_{i,j}^{(\ell)} = 1$ at the k_ℓ^{th} position if a node is assigned to community k_ℓ and zero otherwise. This matrix can be obtained by taking the argument maximum over the rows of community assignment probabilities \mathbf{P}_ℓ for the ℓ^{th} hierarchical layer.

$$\mathbf{C}_\ell = g(\mathbf{P}_\ell) \text{ where } g(\hat{p}_{i,k_\ell}) = \begin{cases} 1 & \text{if node } i \text{ assigned to community } k_\ell \\ 0 & \text{else} \end{cases}$$

For each hierarchical layer ℓ we may compute the community assignment labels. Consider a two-layer hierarchy, the community assignment labels \mathbb{S}_1 from assigning N nodes in the bottom layer to $k_1 < N$ nodes in the super layer is given as

$$\mathbb{S}_1 = \arg \max_{k_1} \mathbf{C}_1$$

more generally, the labels from assigning nodes in layer $\ell - 1$ to layer ℓ is given as:

$$\mathbb{S}_\ell = \arg \max_{k_\ell} \mathbf{C}_\ell$$

• **Compute input to the next classifier:**

Each linear classifier aims to classify the nodes in the previous layer to a subset of nodes which represent the communities of the current layer. For example, the first classifier classifies the N original nodes to k_1 communities and takes as input the embeddings matrix from the auto-encoder. The second classifier which classifies k_1 communities to k_2 communities takes as input the centroids of the k_1 communities in the previous layer which is computed by projecting the embeddings onto the community probabilities matrix \mathbf{P}_1 . These centroids are then activated to ensure regularity denoted by the activation function $\phi(\cdot)$ which may be the identity function:

$$\tilde{\mathbf{X}}^{(1)} = \phi(\mathbf{Z}^T \mathbf{P}_1)^T$$

Thus $\tilde{\mathbf{X}}^{(1)} \in \mathbb{R}^{k_1 \times q}$ is matrix corresponding the centroids of the k_1 predicted communities in q embedded features. In general, for layers $\ell > 1$ in the hierarchy, the input is calculated from the linear combination of the centroids of the previous layer $\ell - 1$ and with the community assignment matrix of the current layer ℓ and activated by function $g_\ell(\cdot)$:

$$\tilde{\mathbf{X}}^{(\ell)} = \phi_\ell \left(\tilde{\mathbf{X}}^{(\ell-1)T} \mathbf{P}_\ell \right)^T$$

Since our main objective is to learn the hierarchical representation of the original graph, we also compute the adjacency matrices corresponding to each hierarchical layer. We want the adjacency matrix for a given hierarchical layer to summarize the connections between and within the communities of that layer. For example, assigning the N original nodes to k_1 communities, the adjacency matrix representing the connections between k_1 super nodes is computed as:

$$\tilde{\mathbf{A}}^{(1)} = \mathbf{P}_1^T \mathbf{A} \mathbf{P}_1$$

In general, the matrix of community assignment probabilities \mathbf{P} which maps the assignments of the nodes in the previous layer to the super nodes (communities) in the current layer can be computed as follows

$$\tilde{\mathbf{A}}^{(\ell)} = \mathbf{P}_\ell^T \tilde{\mathbf{A}}_{\ell-1} \mathbf{P}_\ell$$

The diagonal elements of $\tilde{\mathbf{A}}^{(\ell)}$ are sums of the edge weights for edges between nodes in the same community of the ℓ^{th} hierarchical.

$$\tilde{\mathbf{A}}_{kk}^{(\ell)} = \sum_{i,j \in \mathcal{C}_\ell^{(k)}} a_{ij}$$

Where $\mathcal{C}_\ell^{(k)}$ denotes the set of nodes in the k^{th} community of the ℓ^{th} hierarchical layer. The off diagonal elements of $\tilde{\mathbf{A}}^{(\ell)}$ are the sums of the edge weights between edges connecting nodes in different communities.

$$\tilde{\mathbf{A}}_{k,l}^{(\ell)} = \sum_{v_i \in \mathcal{C}_\ell^{(k)}} \sum_{v_j \in \mathcal{C}_\ell^{(l)}} a_{ij} \quad \forall k \neq l$$

2.2.2 Compute Loss:

For L hierarchical layers

compute the loss function

The total loss function will consist of two primary components: The reconstruction loss \mathcal{L}_R and the modularity loss \mathcal{L}_M

$$L_{\text{Total}} = L_R - \delta L_M$$

where δ denotes a tuning parameter for the modularity component. The primary objective of fitting is to maximize the modularity component while minimizing the reconstruction loss.

$$L_{\text{Total}} = L_{\hat{\mathbf{X}}} + \gamma L_{\hat{\mathbf{A}}} - \delta L_M$$

The reconstruction loss is composed of two reconstruction components.

$$L_R = L_{\hat{\mathbf{X}}} + \gamma L_{\hat{\mathbf{A}}}$$

The first reconstruction component results from reconstruction of the input adjacency matrix \mathbf{A} . Here we employ the binary cross entropy (BCE) loss which compares the input adjacency \mathbf{A} and the reconstructed adjacency matrix $\hat{\mathbf{A}}$ of the bottom level of the hierarchy:

$$L_{\hat{\mathbf{A}}} = \frac{1}{\sum_i \sum_j a_{ij}} \sum_{a_{ij} \in \mathbf{A}} -(a_{ij} \cdot \log(\hat{a}_{ij}) + (1 - a_{ij}) \cdot \log(1 - \hat{a}_{ij}))$$

This component of the reconstruction loss aims to ensure that the graph autoencoder maintains the structure of the original input graph.

The second reconstruction component is the squared error loss between the input and reconstructed node attributes:

$$L_{\hat{\mathbf{X}}} = \|\mathbf{X} - \hat{\mathbf{X}}\|_F = \sum_i^N \sum_j^p (x_{ij} - \hat{x}_{ij})^2$$

where F denotes the Frobenius norm (element wise MSE). This loss prioritizes the reconstruction of the original node features/attributes.

The second component of the total loss function is the modularity loss L_M . This component aims to maximize the modularity of the communities in each hierarchical layer. Therefore, this component is represented by the sum of the modularity of the ℓ hierarchical communities represented by the adjacency matrices $\tilde{\mathbf{A}}^{(\ell)}$

$$L_M = \sum_{i=1}^{\ell} L_i = \sum_{i=1}^{\ell} \frac{1}{4n_{\ell}} \text{Tr}(\mathbf{P}_{\ell}^T \mathbf{B}_{\ell} \mathbf{P}_{\ell})$$

where \mathbf{P}_{ℓ} is the matrix of community assignment probability for the ℓ^{th} hierarchical layer, q_{ℓ} is the total number of edges in the graph for the ℓ^{th} hierarchical layer, $\text{Tr}(\cdot)$ denotes the trace function, and \mathbf{B}_{ℓ} is the modularity matrix for the ℓ^{th} hierarchical layer defined as:

$$\mathbf{B}_{\ell} = \tilde{\mathbf{A}}_{i,j}^{(\ell)} - \frac{d(v_i) \cdot d(v_j)}{2n_{\ell}}$$

$d(\cdot)$ is a function which returns the degree of a node. A linear formulation of the modularity matrix can be computed as follows

$$\mathbf{B}_{\ell} = \tilde{\mathbf{A}}^{(\ell)} - \frac{1}{2n_{\ell}} \mathbf{r} \otimes \mathbf{r}$$

where \otimes denotes the outer product of two vectors, $\mathbf{r} \in \mathbb{R}^{k_{\ell}}$ is a vector of the node degrees found via the row summation $\mathbf{r} = \tilde{\mathbf{A}}^{(\ell)} \mathbf{1}_{k_{\ell}}$. n_{ℓ} is the total number of edges in graph:

$$n_{\ell} = \frac{1}{2} \sum_i^{k_{\ell}} \sum_j^{k_{\ell}} \tilde{\mathbf{A}}_{ij}^{(\ell)}$$

2.2.3 Compute backward pass:

For all $\omega_i \in \Omega$

Back-propagate to find gradients

$$\nabla_{\omega_i} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \omega_i} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \dots \frac{\partial f_n}{\partial \omega_i}$$

Update all parameters

$$\omega_i^{(t+1)} \leftarrow \omega_i^{(t)} - g(\nabla_{\omega_i} \mathcal{L})$$

Figure 1: Proposed HRGN model. Example represents a model constructed for a hierarchy with $\ell = 2$ super layers.

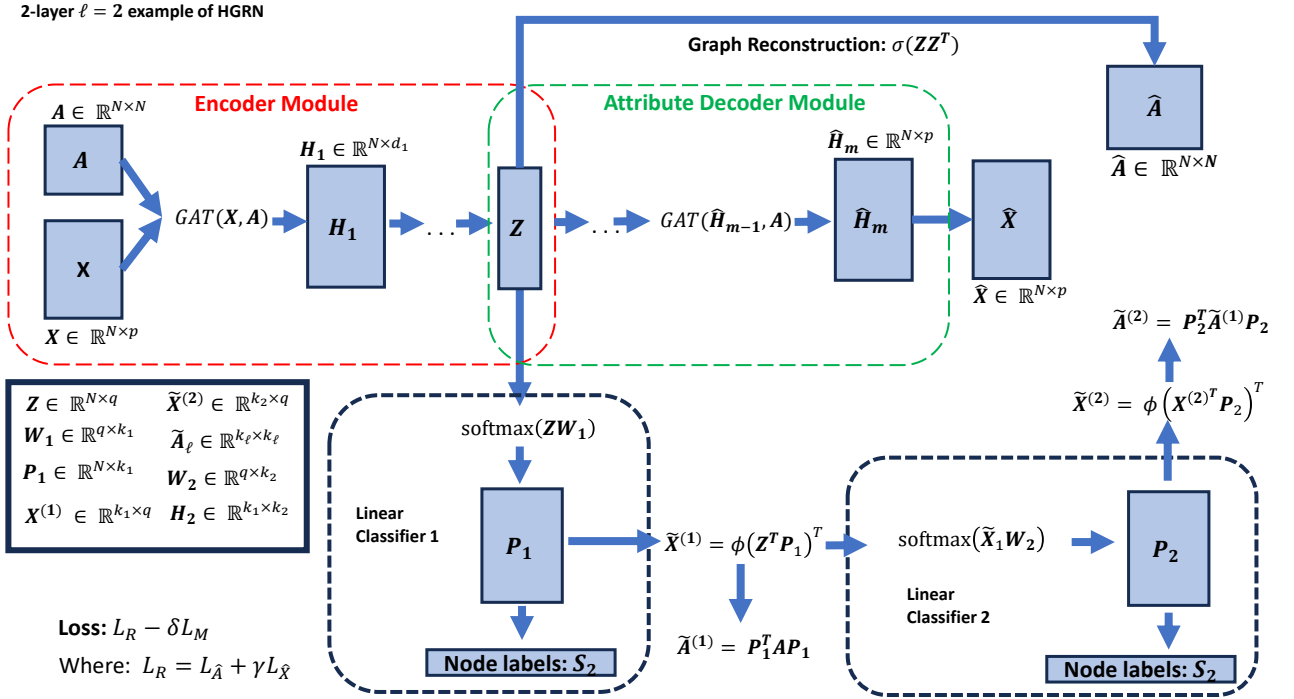


Table 1: Notation and explanations

Symbol	Dimension	Explanation
ℓ		The number of super layers in the hierarchy
L		The total number of hierarchical layers
N		The number of nodes input graph \mathbf{A}
v_i		the i^{th} node in the graph
p		The number of attributes in node attribute matrix \mathbf{X}
m		The number of hidden encoder/decoder layers in GATE
d_m		The column dimension of the m^{th} hidden layer of the encoder/decoder
k_ℓ		The number of nodes (i.e communities) in the ℓ^{th} super layer
$d(\cdot)$		A function which returns the degree of a node
n_ℓ		The number of edges in ℓ^{th} super layer network \mathcal{G}_ℓ
\mathbf{A}	$\in \mathbb{R}^{N \times N}$	The input adjacency matrix
\mathbf{X}	$\in \mathbb{R}^{N \times p}$	The input node-attribute matrix
\mathbf{H}_m	$\in \mathbb{R}^{N \times d_m}$	The representation of the nodes in $m - 1^{th}$ hidden layer of the encoder
$\hat{\mathbf{H}}_m$	$\in \mathbb{R}^{N \times d_m}$	The representation of the nodes in $m - 1^{th}$ hidden layer of the decoder
\mathbf{W}_m	$\in \mathbb{R}^{d_{m-1} \times d_m}$	the weights corresponding to the m^{th} hidden layer of the encoder module
$\hat{\mathbf{W}}_m$	$\in \mathbb{R}^{d_m \times d_{m-1}}$	the weights corresponding to the $m - 1^{th}$ hidden layer of the decoder
\mathbf{Z}	$\in \mathbb{R}^{N \times q}$	the embedding matrix
\mathbf{P}_ℓ	$\in \mathbb{R}^{k_{\ell-1} \times k_\ell}$	The matrix of assignment probabilities of the ℓ^{th} hierarchical (super) layer
$\mathcal{C}_\ell^{(k)}$		The set of nodes in the k^{th} community of the ℓ^{th} hierarchical layer
$\tilde{\mathbf{X}}^{(\ell)}$	$\in \mathbb{R}^{k_\ell \times q}$	The centroids of the communities in the ℓ^{th} hierarchical layer
$\tilde{\mathbf{A}}^{(\ell)}$	$\in \mathbb{R}^{k_\ell \times k_\ell}$	The adjacency matrix corresponding to the ℓ^{th} hierarchical layer
\mathbf{B}_ℓ	$\in \mathbb{R}^{k_\ell \times k_\ell}$	The modularity matrix of $\tilde{\mathbf{A}}^{(\ell)}$

References

- [1] Amin Salehi and Hasan Davulcu. Graph attention auto-encoders. *arXiv preprint arXiv:1905.10715*, 2019.
- [2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
- [3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [4] Xinchuang Zhou, Lingtao Su, Xiangju Li, Zhongying Zhao, and Chao Li. Community detection based on unsupervised attributed network embedding. *Expert Systems with Applications*, 213:118937, 2023.
- [5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.