Network Representation Learning a.k.a. Graph Embedding

Bogumił Kamiński
Paweł Prałat
François Théberge*
theberge@ieee.org

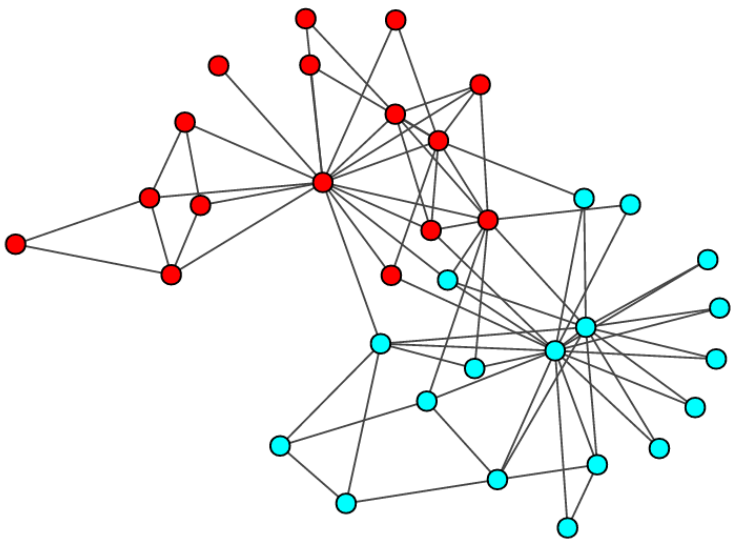
August 2019

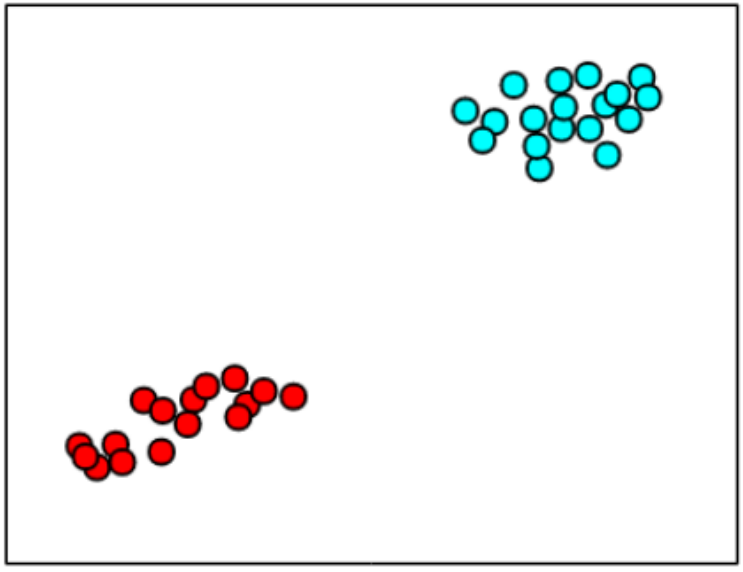
Outline

1. Quick overview of graph embedding
2. A few algorithms: node2vec, LINE, Verse
3. Framework to compare embeddings
4. Examples

Graph Embedding

Objective: Map network (nodes) to vector (feature) space





Graph Embedding

Map *similar* nodes to nearby location in vector space.
Similar may have different meaning:

- close w.r.t. graph topology
- similar role in graph (ex: similar degree)
- similar node attributes

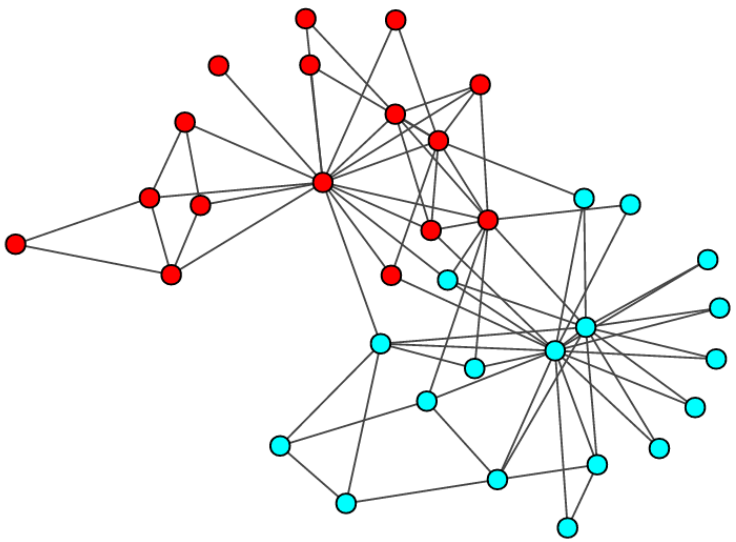
Graph Embedding

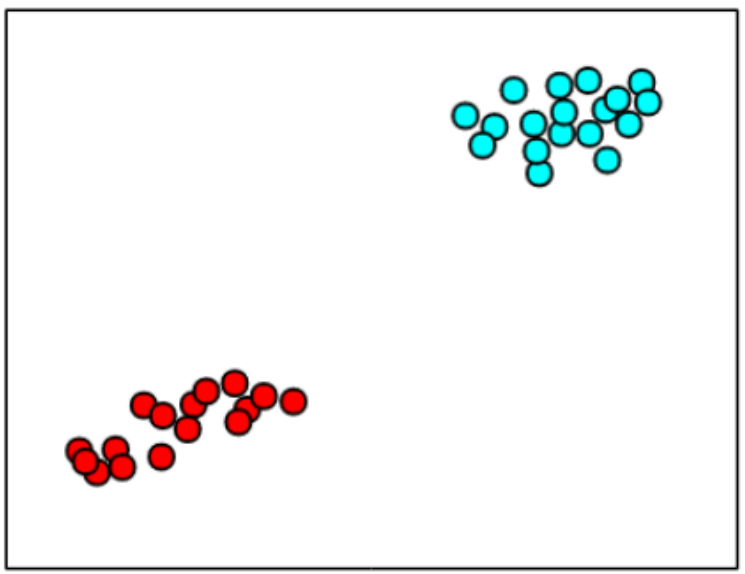
Examples of applications:

- feature learning (rather than engineering)
- visualization
- link prediction
- community detection
- anomaly detection
- network evolution (dynamics)

Graph Embedding

INPUT: $G = (V, E)$
OUTPUT: vectors $z_v \in \mathbb{R}^k, \ \forall v \in V$



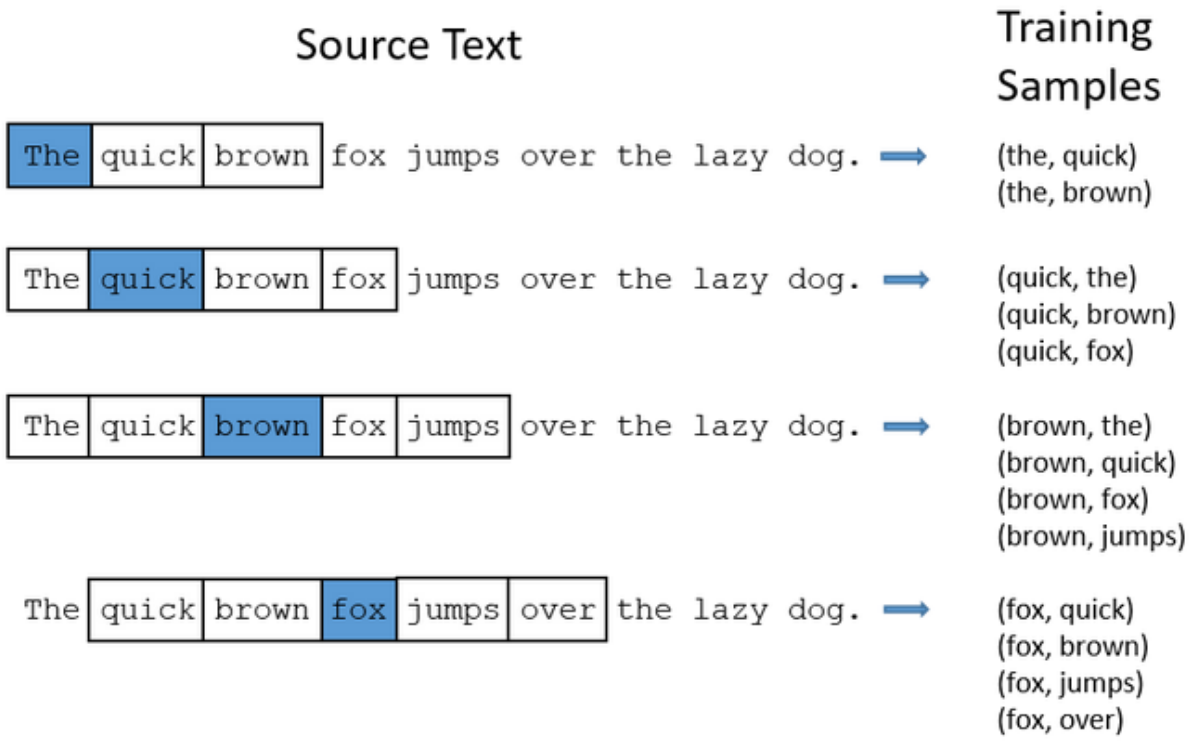


Graph Embedding

Several methods are based on random walks and the SkipGram approach for word embedding.

- A word can be characterized by the company it keeps
- Words in similar context (nearby word(s)) have similar meaning
- Consider a window around each word; build ”word vectors” (ex: word2vec)
- Use those as training data

SkipGram



ref: www.mccarmichael.com

DeepWalk

- words are nodes $v \in V$
- sentences are random walks on G
- word frequency in sentences show power law distribution
- vertex distribution in walks also shows power law

node2vec

- node2vec defines biased random walks
- mix of breadth and depth first search

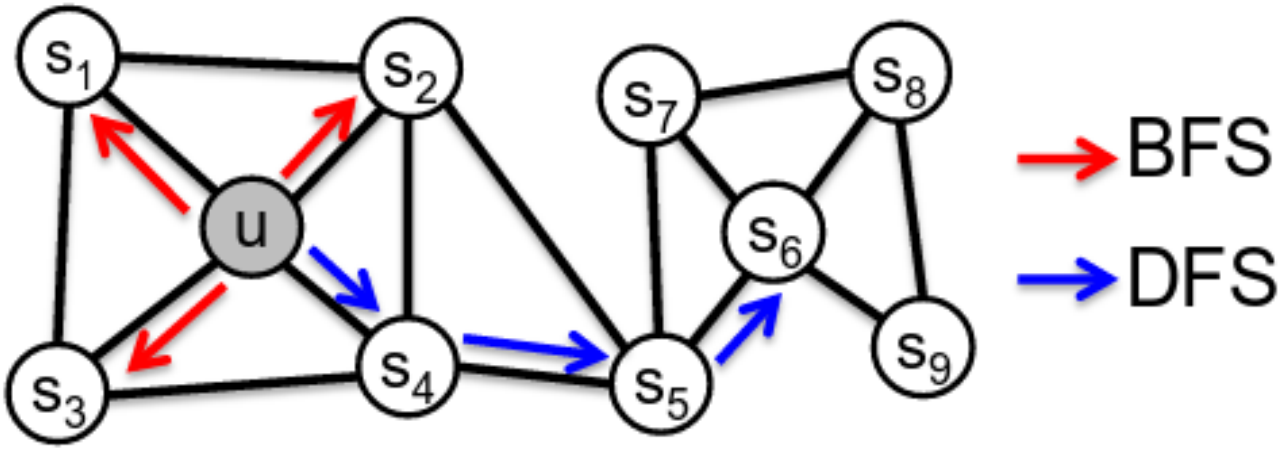


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

REF: node2vec: Scalable Feature Learning for Networks, A. Grover and J. Leskovec

node2vec

Key parameters:

- *p*: controls probability of re-visiting same node (stay in neighbourhood)
- *q*: controls probability of exploring further away

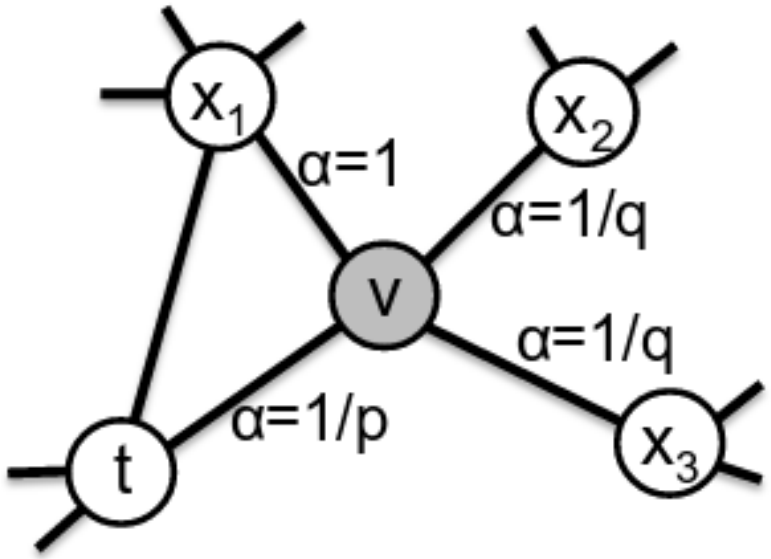


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

node2vec

Parameters allow to tradeoff between:

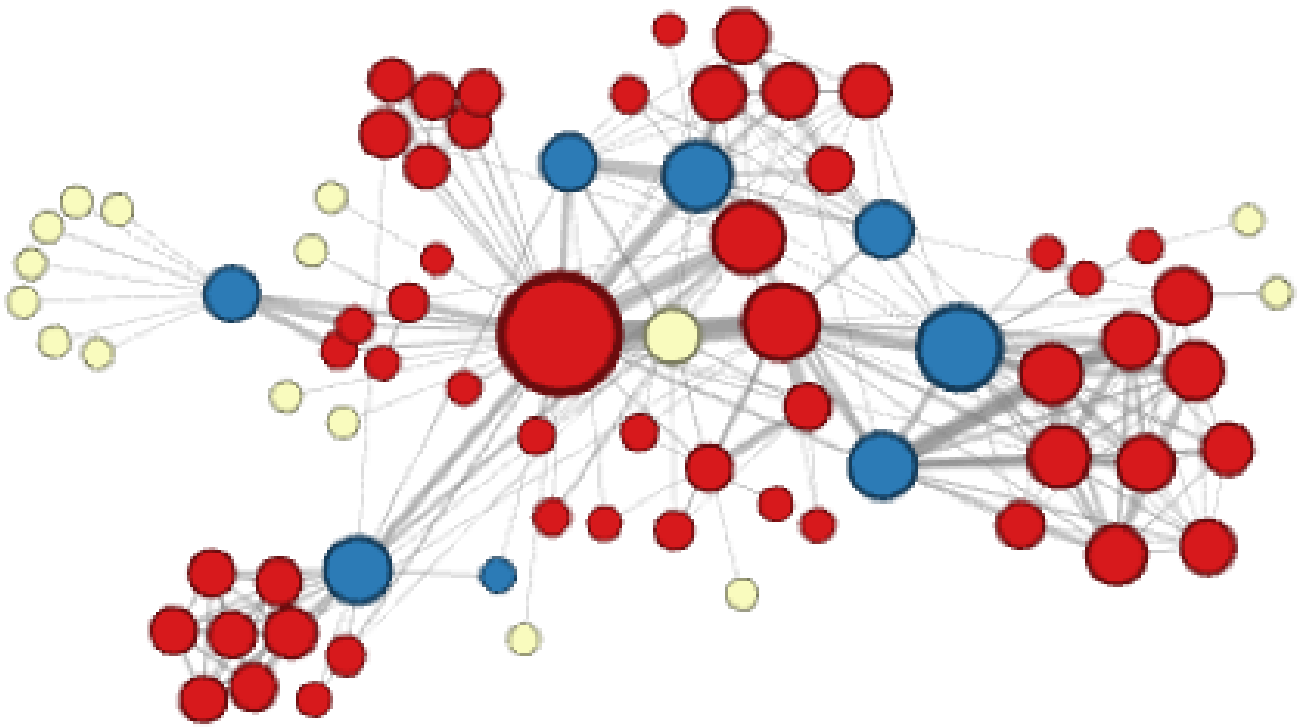
- low p: explore locally; this will focus on community structure in the topology of the graph (homophily);
- low q: explore further away; this allows to capture some structural similarity between nodes (ex: hubs, bridges);

node2vec

Example from Les Misérables co-occurrence graph showing homophily:

node2vec

Example from Les Misérables co-occurrence graph structural equivalence:



Other Algorithms

Category	Year	Published	Method	Time Complexity	Properties preserved
Factorization	2000	Science[26]	LLE	$O(\|E\|d^2)$	1^{st} order proximity
	2001	NIPS[25]	Laplacian Eigenmaps	$O(\|E\|d^2)$	
	2013	WWW[21]	Graph Factorization	$O(\|E\|d)$	
	2015	CIKM[27]	GraRep	$O(\|V\|^3)$	$1-k^{th}$ order proximities
	2016	KDD[24]	HOPE	$O(\|E\|d^2)$	
Random Walk	2014	KDD[28]	DeepWalk	$O(\|V\|d)$	
	2016	KDD[29]	*node2vec*	$O(\|V\|d)$	$1-k^{th}$ order proximities, structural equivalence
Deep Learning	2016	KDD[23]	SDNE	$O(\|V\|\|E\|)$	1^{st} and 2^{nd} order proximities
	2016	AAAI[30]	DNGR	$O(\|V\|^2)$	$1-k^{th}$ order proximities
	2017	ICLR[31]	GCN	$O(\|E\|d^2)$	$1-k^{th}$ order proximities
Miscellaneous	2015	WWW[22]	LINE	$O(\|E\|d)$	1^{st} and 2^{nd} order proximities

Table 1: List of graph embedding approaches

REF: Graph Embedding Techniques, Applications, and Performance: A Survey, P. Goyal and E. Ferrara

Embedding Algorithms

In our tests, we used:

- node2vec with $q = 1$ and varying p (`https://snap.stanford.edu/node2vec/`);
- VERSE: Versatile Graph Embeddings from Similarity Measures (with personalized page rank) with default parameters (`https://github.com/xgfs/verse`);
- LINE: Large-scale Information Network Embedding, which uses an approximate factorization of the adjacency matrix trying to preserve first and second order proximities (`https://github.com/tangjianpku/LINE`).

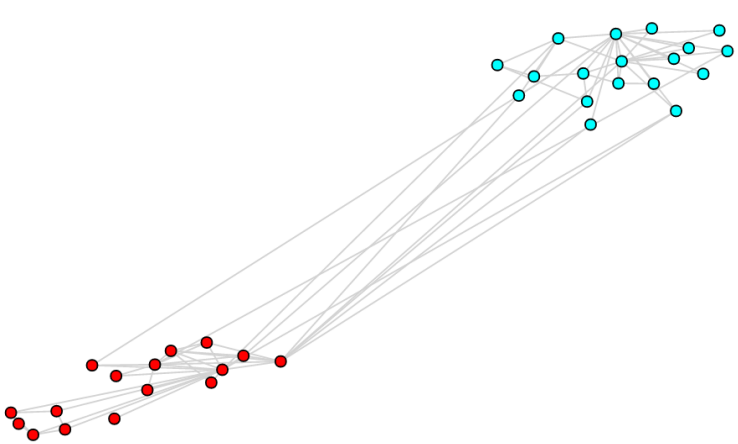
Which embedding?

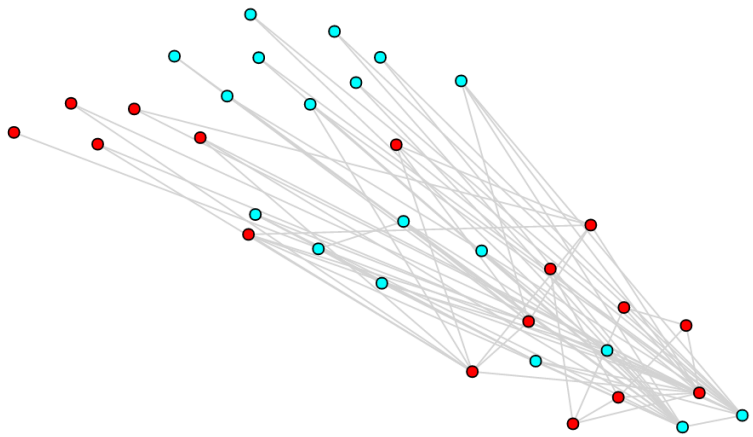
- which embedding algorithm should I use?
- how to I select the parameters?
- how do I know if the representation is good?
- GIGO: bad representation in vector space leads to bad results ...

Which embedding?

Results can vary a lot between algorithms, and with the choice of parameters...

Ex: two instances of node2vec on the Karate dataset:





Proposed Framework

Given $G = (V, E)$ on n vertices with the degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$ and an embedding of its vertices to k-dimensional space, $\mathcal{E} : V \to \mathbb{R}^k$.

Our goal is to assign a "divergence score" to this embedding.

The lower the score, the better the embedding is. This will allow us to compare several embeddings, possibly in different dimensions.

Overview

- Non-random graphs exhibit community-like structure
- We group nodes in clusters
- We measure edge density between and within clusters
- We compare with predicted densities from a spatial model in embedded (vector) space by computing a divergence score
- We select the embedding with the best score

Overview

We have two main ingredients in our framework:

- **Graph topology view:** a good, stable **graph clustering algorithm**; we use ECG by default, but we also tried with Louvain and InfoMap;
- **Spatial view:** we introduce the **Geometric Chung-Lu (GCL)** model based on the degree distribution **w** and the embedding $\mathcal{E}$.

Chung-Lu Model

In the original Chung-Lu model, each set $e = \{v_i, v_j\}$, $v_i, v_j \in V$, is independently sampled as an edge with probability given by:

$$p_{i,j} = \begin{cases} \frac{\deg_G(v_i)\deg_G(v_j)}{2|E|}, & i \neq j \\ \frac{\deg_G^2(v_i)}{4|E|}, & i = j. \end{cases}$$

It yields a distribution that preserves the expected degree for each vertex.

Geometric Chung-Lu (GCL) Model

We now consider the expected degree distribution

$$\mathbf{w} = (w_1, \ldots, w_n) = (\deg_G(v_1), \ldots, \deg_G(v_n))$$

as well as an embedding of the nodes $\mathcal{E} : V \to \mathbb{R}^k$ such that we know all distances:

$$d_{i,j} = \mathrm{dist}(\mathcal{E}(v_i), \mathcal{E}(v_j)).$$

Model should be such that $p_{i,j} \propto g(d_{i,j})$ for some decreasing function g, so long edges should occur less frequently than short ones.

Geometric Chung-Lu (GCL) Model

There are many natural choices for $g()$. We use the following, normalized function $g : [0, \infty) \to [0, 1]$ for a fixed $\alpha \in [0, \infty)$:

$$g(d) := \left(1 - \frac{d - d_{\min}}{d_{\max} - d_{\min}}\right)^{\alpha},$$

where

$$\begin{aligned} d_{\min} &= \min\{\mathrm{dist}(\mathcal{E}(v), \mathcal{E}(w)) : v, w \in V\} \\ d_{\max} &= \max\{\mathrm{dist}(\mathcal{E}(v), \mathcal{E}(w)) : v, w \in V\} \end{aligned}$$

We use clipping to force $g(d_{\min}) < 1$ and/or $g(d_{\max}) > 0$.

Geometric Chung-Lu (GCL) Model

With $\alpha = 0$, we recover the original Chung-Lu model, so pairwise distances are neglected.

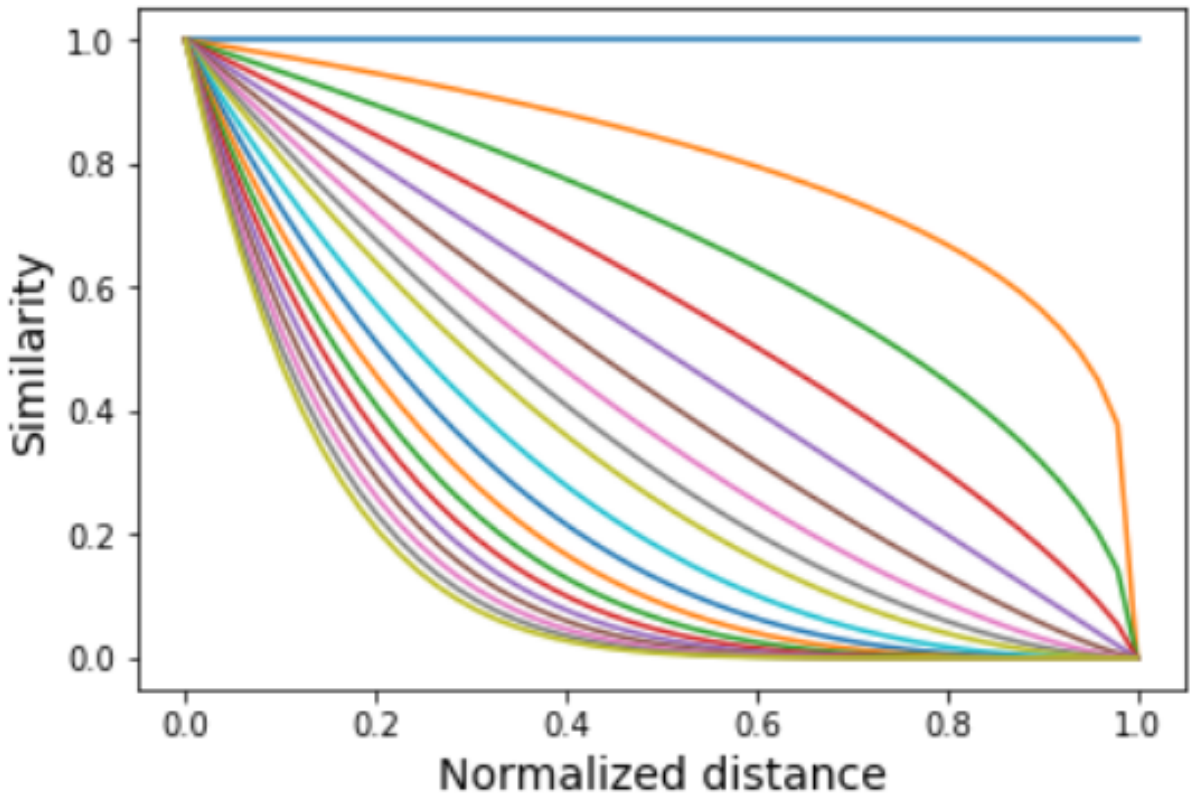
The larger parameter α is, the larger aversion for long edges is.

Thus, the only parameter of the model is $\alpha \in [0, \infty)$.

In practice, we try a range of values and keep the best fit.

Geometric Chung-Lu (GCL) Model

Decreasing function $g()$ for $0 \leq \alpha \leq 7$:



Geometric Chung-Lu (GCL) Model

The GCL model is the random graph $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$ on the vertex set $V = \{v_1, \ldots, v_n\}$ where v_i, v_j, forms an edge with probability:

$$p_{i,j} = x_i x_j g(d_{i,j})$$

for some carefully tuned weights $x_i \in \mathbb{R}_+$.

The weights are selected such that the expected degree of v_i is w_i for all i:

$$w_i = \sum_j p_{i,j} = x_i \sum_j x_j g(d_{i,j}).$$

Geometric Chung-Lu (GCL) Model

Theorem

There exists a unique selection of weights, provided that the maximum degree in G is less than the sum of degrees of all other vertices.

Since each connected component of G can be embedded independently, we may assume that G is connected and so the minimum degree of G is at least 1. As a result, this very mild condition is trivially satisfied unless G is a star on n vertices.

Solving GCL

We use a simple numerical approximation procedure.

Start with an arbitrary vector $\mathbf{t}^0 = (t_1^0, \ldots, t_n^0) = (1, \ldots, 1)$

Given $\mathbf{t}^s = (t_1^s, \ldots, t_n^s)$, if we introduce an edge between v_i and v_j with probability

$$p_{i,j}^s = t_i^s t_j^s g(d_{i,j}),$$

then the expected degree of v_i would be

$$s_i^s = \sum_j p_{i,j}^s = t_i^s \sum_j t_j^s g(d_{i,j}).$$

Adjust the weights so that s_i^s matches w_i by replacing t_i^s with $t_i^s(w_i/s_i^s)$.

Solving GCL

This also affect other values of $\mathbf{s}^s$ and changes in other parts of $\mathbf{t}$ affect s_i^s too.

Thus, we let each vertex take a small step into the right direction

This process quickly converge to the desired state: s_i^s being very close to w_i for all i.

Solving GCL

For each i, $1 \leq i \leq n$, we define

$$t_i^{s+1} = (1-\epsilon)t_i^s + \epsilon t_i^s(w_i/s_i^s) = t_i^s + \epsilon t_i^s(w_i/s_i^s - 1).$$

Repeat the tuning process until $\max_i |\mathbf{w}_i - s_i^s| < \delta$.

We used $\epsilon = 0.1$ and $\delta = 0.001$.

Algorithm

Algorithm to compute embedding divergence score

Given $G = (V, E)$, its degree distribution $\mathbf{w}$ on V, and an embedding $\mathcal{E} : V \to \mathbb{R}^k$ of its vertices we perform the five steps detailed next.

We obtain $\Delta_{\mathcal{E}}(G)$, a *divergence score* for the embedding.

We can apply this algorithm to compare several embeddings $\mathcal{E}_1, \ldots, \mathcal{E}_m$, and select the best one.

Algorithm Step 1

Run some stable *graph* clustering algorithm on G to obtain a partition $\mathbf{C}$ of the vertex set V into ℓ communities $C_1, \ldots, C_\ell$.

We use ECG by default, but any good algorithm will do.

Algorithm Step 2

Let:
c_i: proportion of edges with both endpoints in C_i
$c_{i,j}$: proportion of edges with one endpoint in C_i and the other one in C_j

Define:

$$\bar{\mathbf{c}} = (c_{1,2}, \ldots, c_{1,\ell}, c_{2,3}, \ldots, c_{2,\ell}, \ldots, c_{\ell-1,\ell}), \ \hat{\mathbf{c}} = (c_1, \ldots, c_\ell)$$

These *graph vectors* characterize partition $\mathbf{C}$ from the perspective of G.

The embedding $\mathcal{E}$ does *not* affect the vectors $\bar{\mathbf{c}}$ and $\hat{\mathbf{c}}$.

Algorithm Step 3

We repeat steps 3-4 over a range of values for α

Given $\alpha \in \mathbb{R}_+$, consider $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$, the GCL model.

From this model, we compute:
b_i: expected proportion of edges within C_i
$b_{i,j}$: expected proportion of edges with one endpoint in C_i and the other one in C_j

We get:

$$\bar{\mathbf{b}}_{\mathcal{E}}(\alpha) = (b_{1,2}, \ldots, b_{1,\ell}, b_{2,3}, \ldots, b_{2,\ell}, \ldots, b_{\ell-1,\ell})$$

$$\hat{\mathbf{b}}_{\mathcal{E}}(\alpha) = (b_1, \ldots, b_\ell)$$

These vectors characterizes partition $\mathbf{C}$ from the perspective of the embedding $\mathcal{E}$.

Algorithm Step 4

We compute the distances between $\bar{\mathbf{c}}$ and $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$, and between $\hat{\mathbf{c}}$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$.

We use the Jensen-Shannon divergence (JSD):

$$\Delta_\alpha = \frac{1}{2} \cdot \left(JSD(\bar{\mathbf{c}}, \bar{\mathbf{b}}(\alpha)) + JSD(\hat{\mathbf{c}}, \hat{\mathbf{b}}(\alpha)) \right).$$

This is the (divergence) score for a given α.

Algorithm Step 5

From repeated steps 3-4, we have a collection of scores Δ_α.

We select $\hat{\alpha} = \text{argmin}_\alpha \, \Delta_\alpha$

Define the *divergence score* for embedding $\mathcal{E}$ on G as:

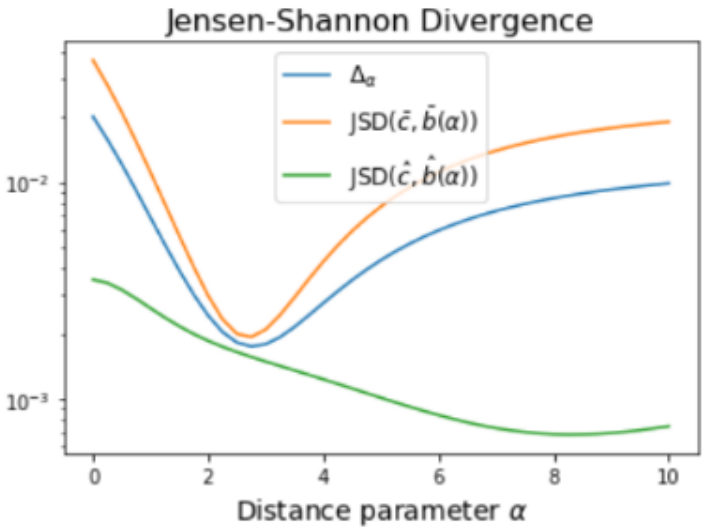
$$\Delta_{\mathcal{E}}(G) = \Delta_{\hat{\alpha}}$$

Algorithm

To compare several embeddings for the same graph G, we repeat steps 3-5 above and compare the divergence scores (lower score is better).

Steps 1-2 are done only once, so we use the same partition into ℓ communities for each embedding.

Karate Club Dataset



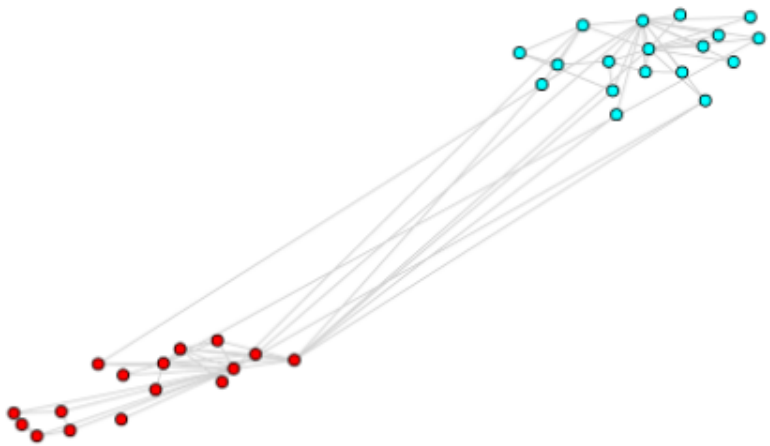
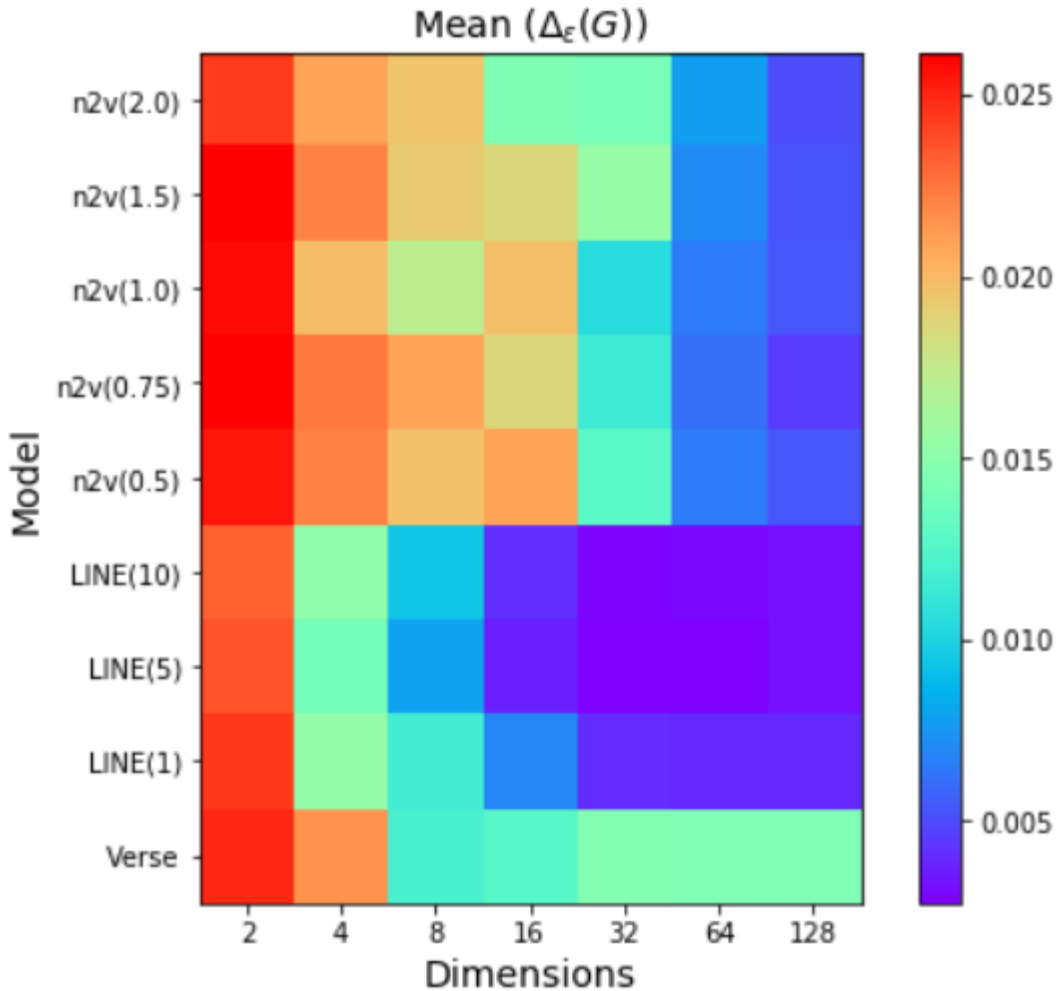
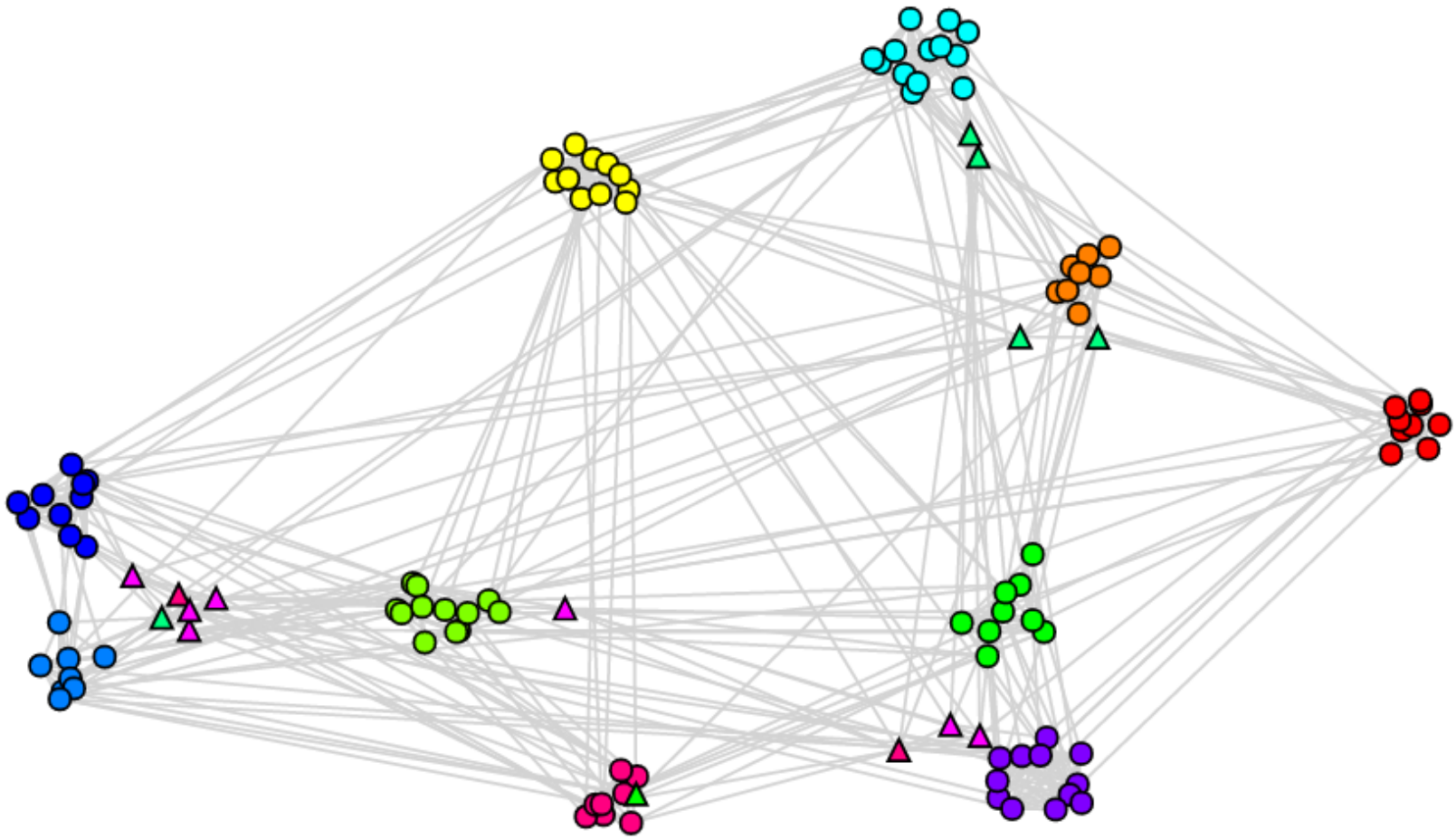


Figure 1: The Karate Club Graph. We illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). The colors represent the two ground-truth communities.

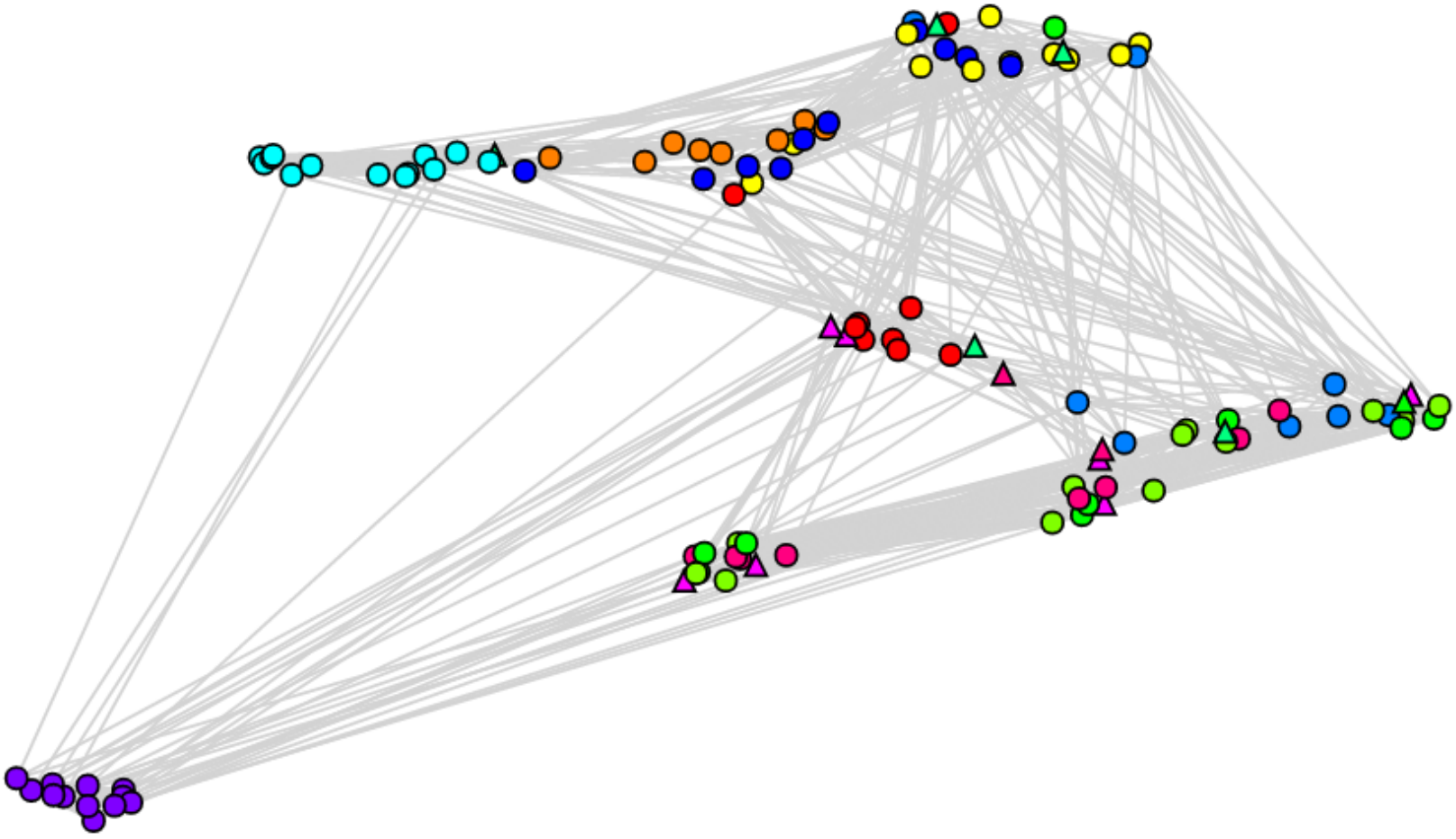
Football Dataset - 630 embeddings



Football Dataset - best



Football Dataset- worst



Football Dataset- Other Graph Clustering Algrithms

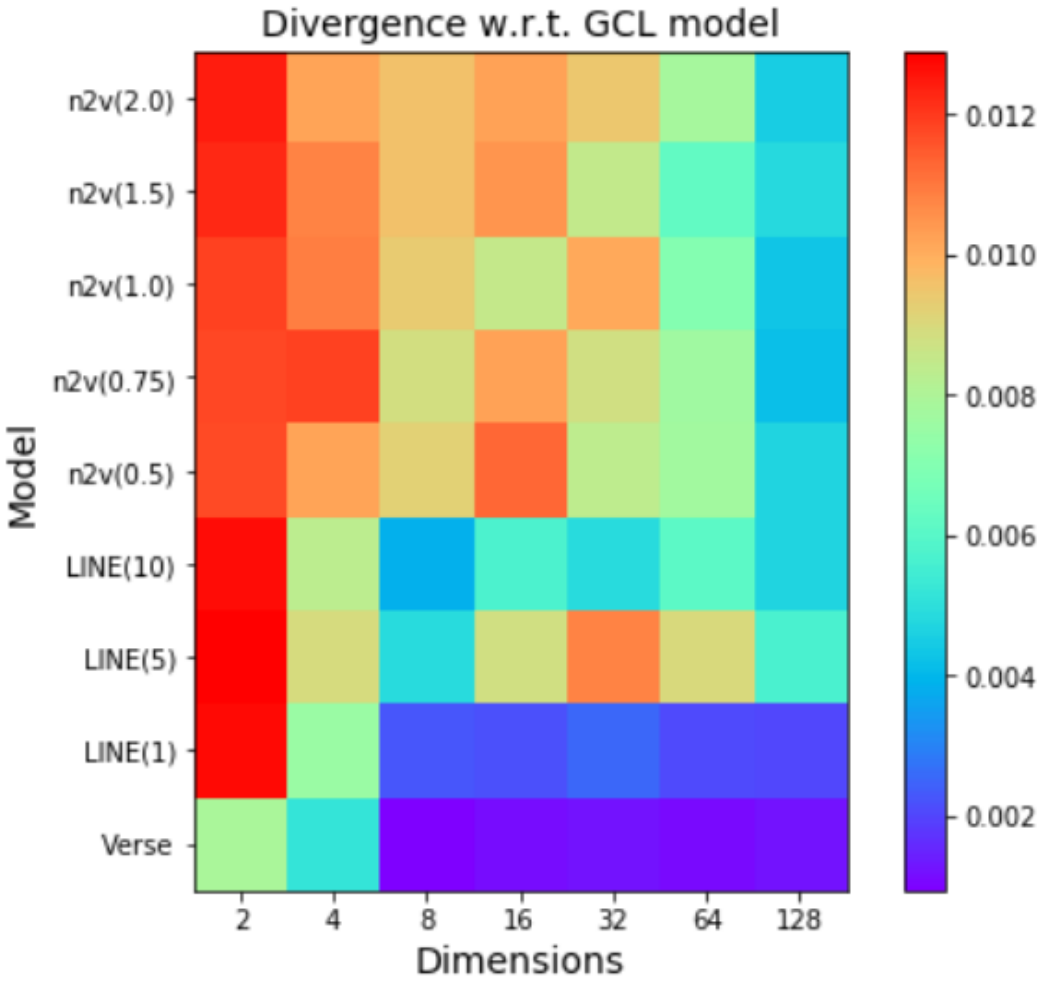
We re-ran the whole experiment using respectively the Louvain and InfoMap algorithms for clustering the Football graph.

Ranked embeddings are well correlated.

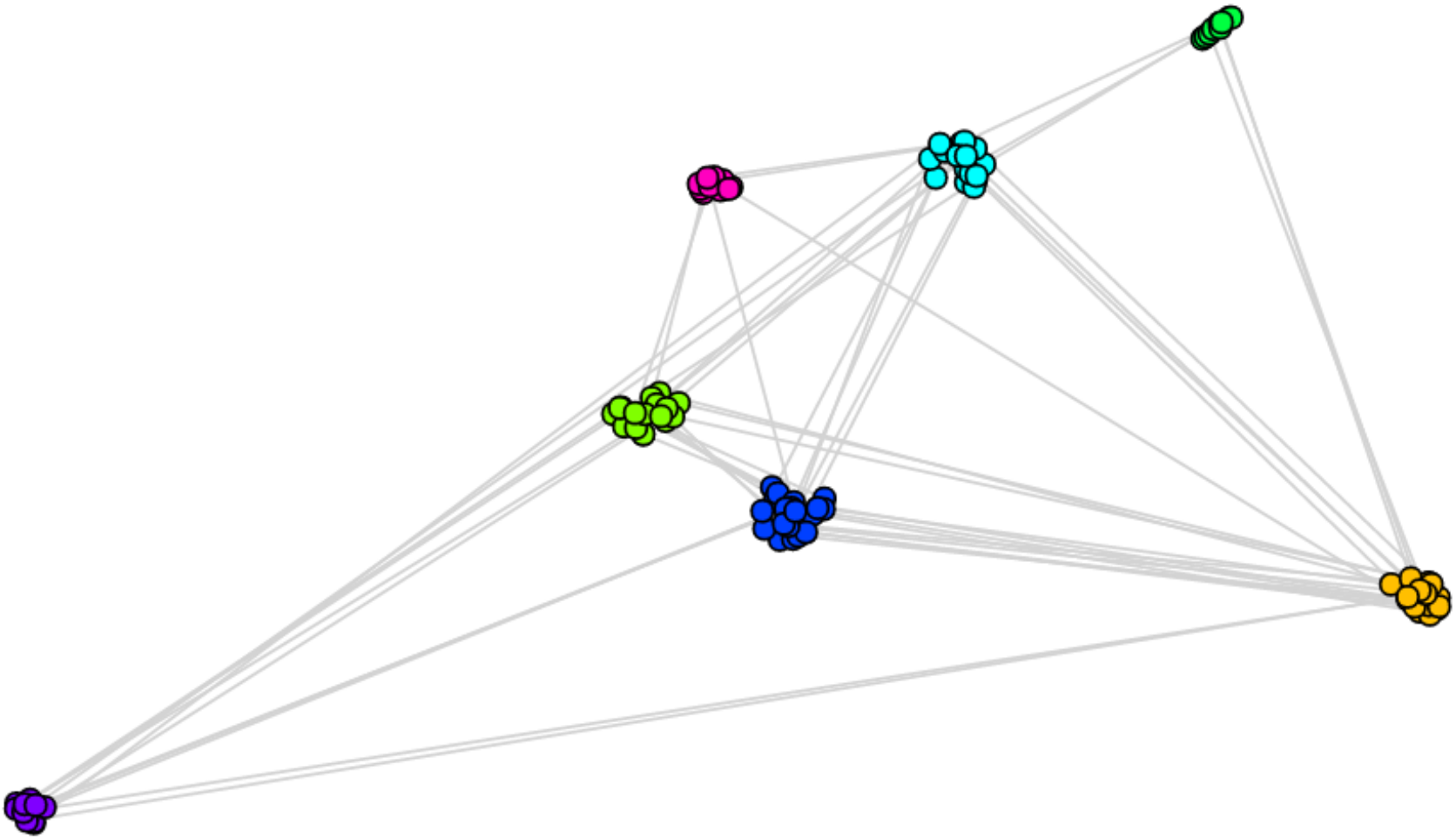
Algorithms	ECG	Louvain
Louvain	0.81	
InfoMap	0.83	0.79

Table 1: Kendall-tau correlation between all ranked embeddings on the College Football graph using 3 different graph clustering algorithms.

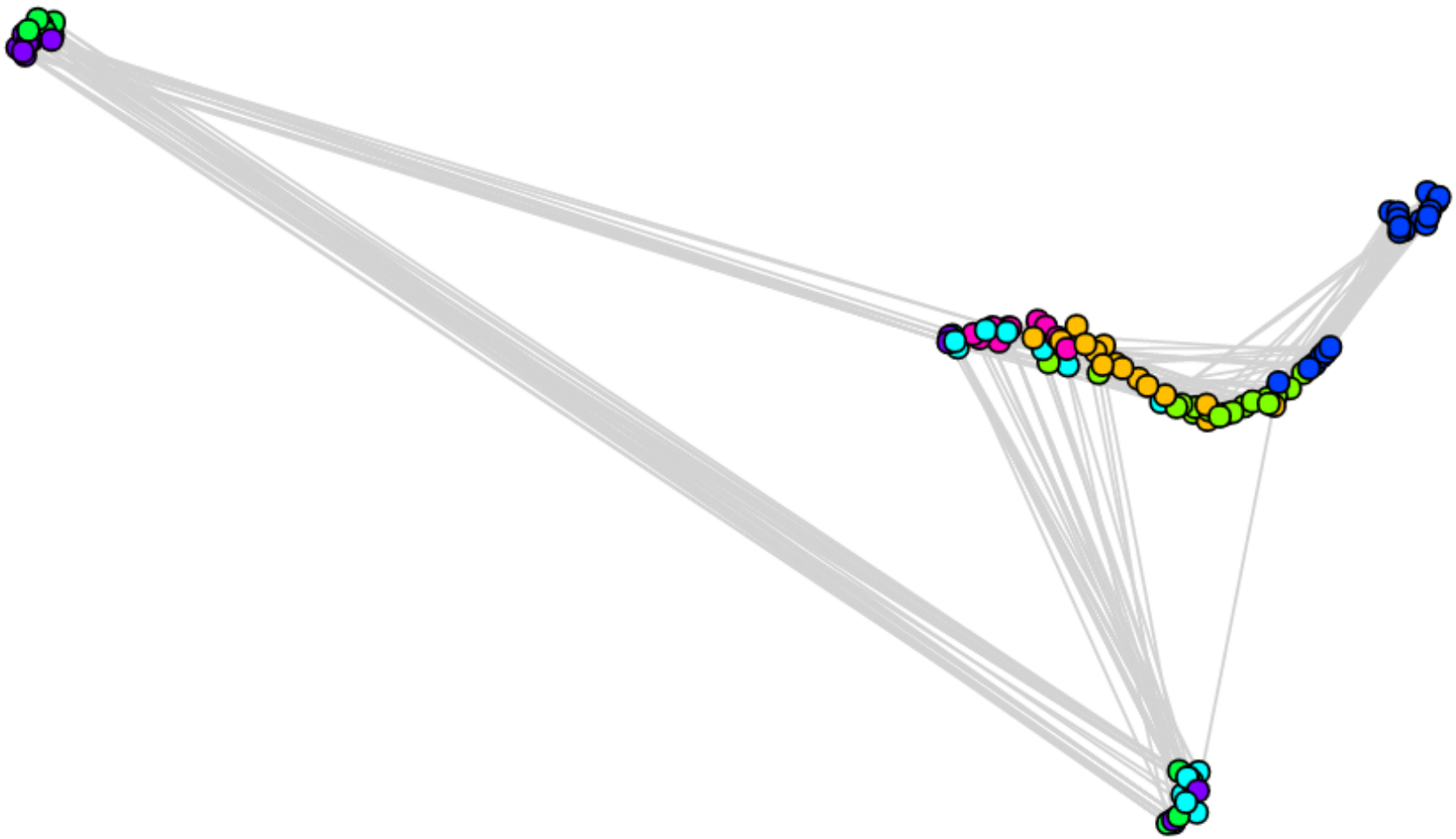
LFR15 Dataset - 630 embeddings



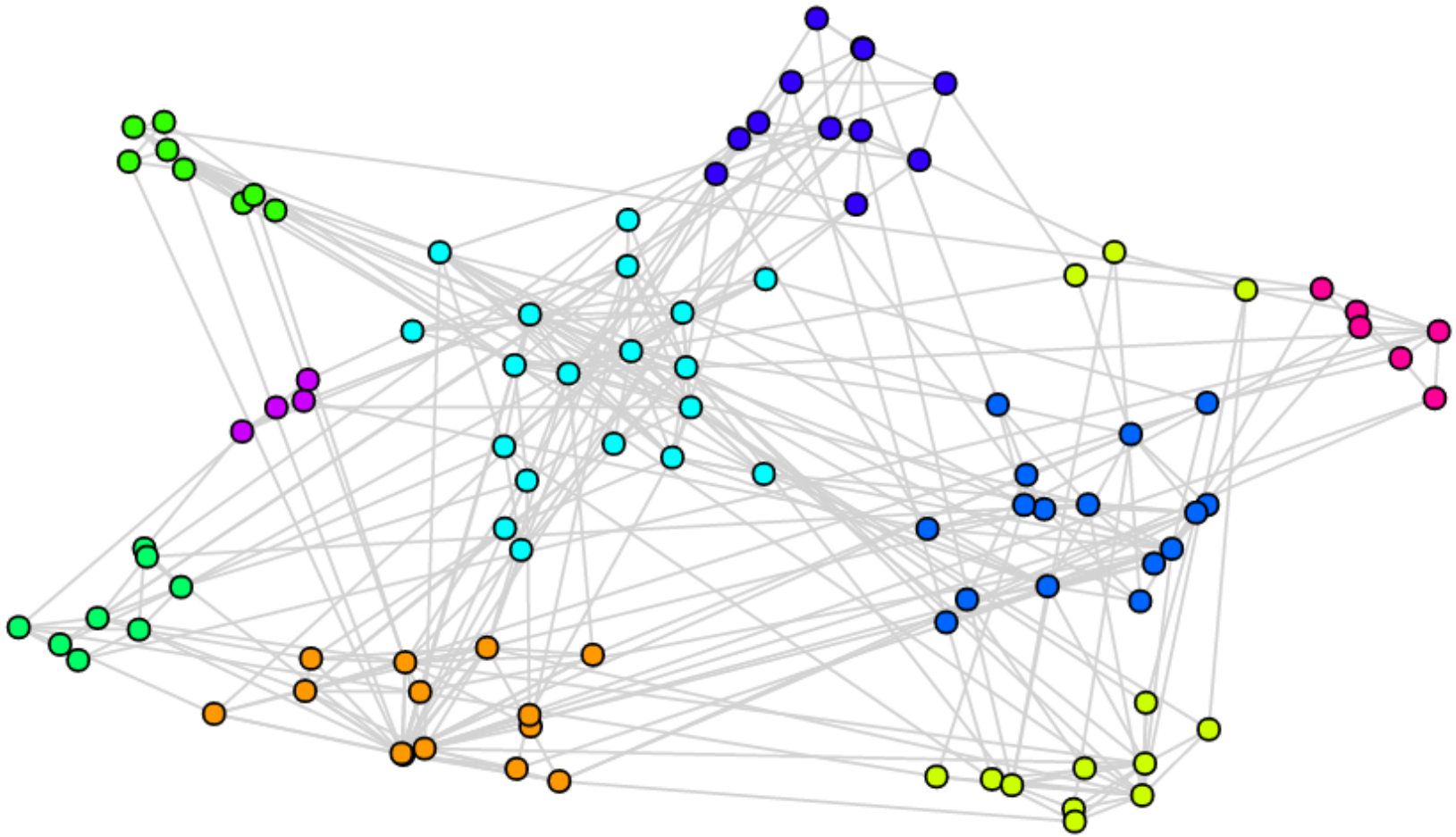
LFR15 - best



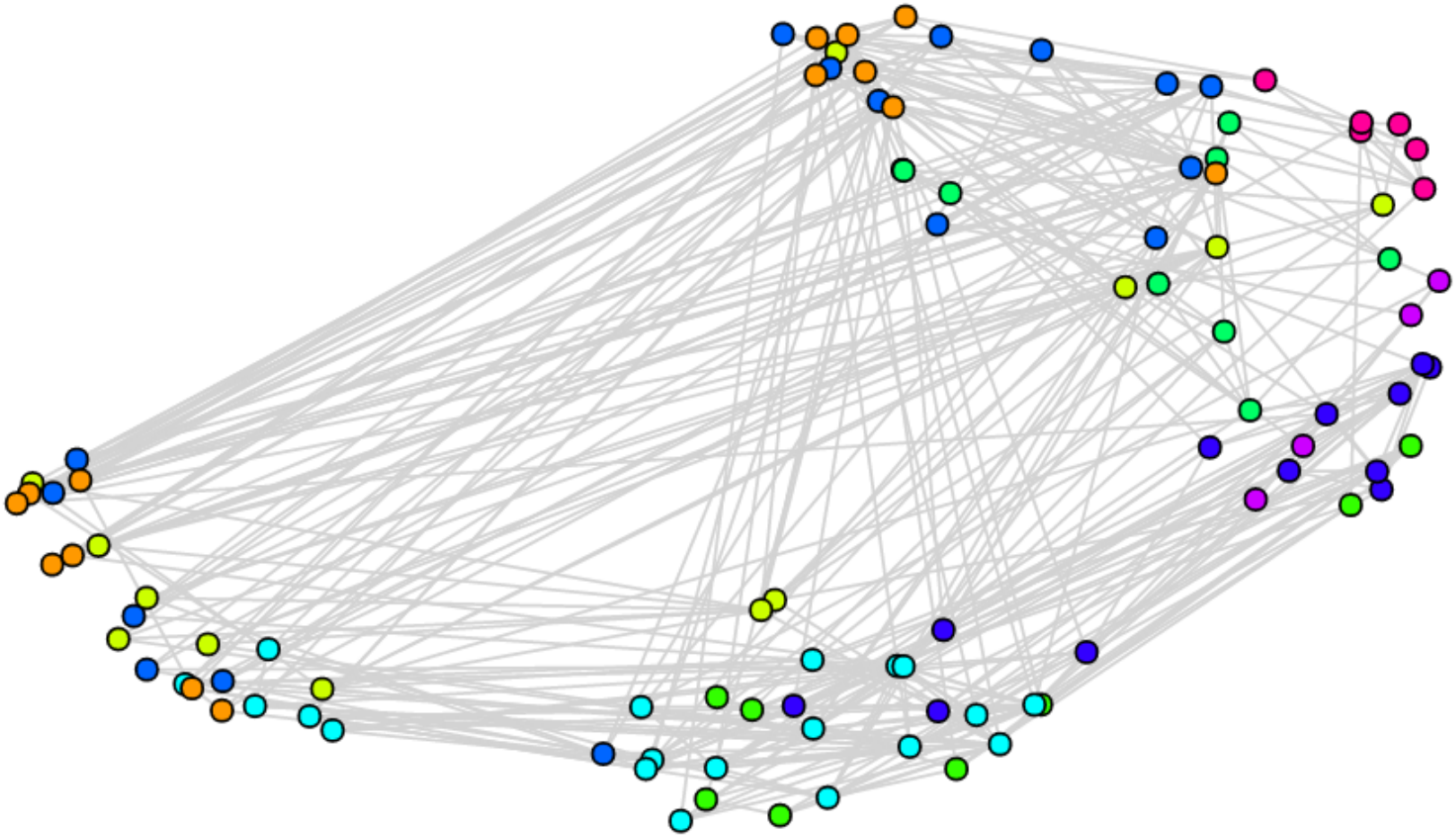
LFR15 - worst



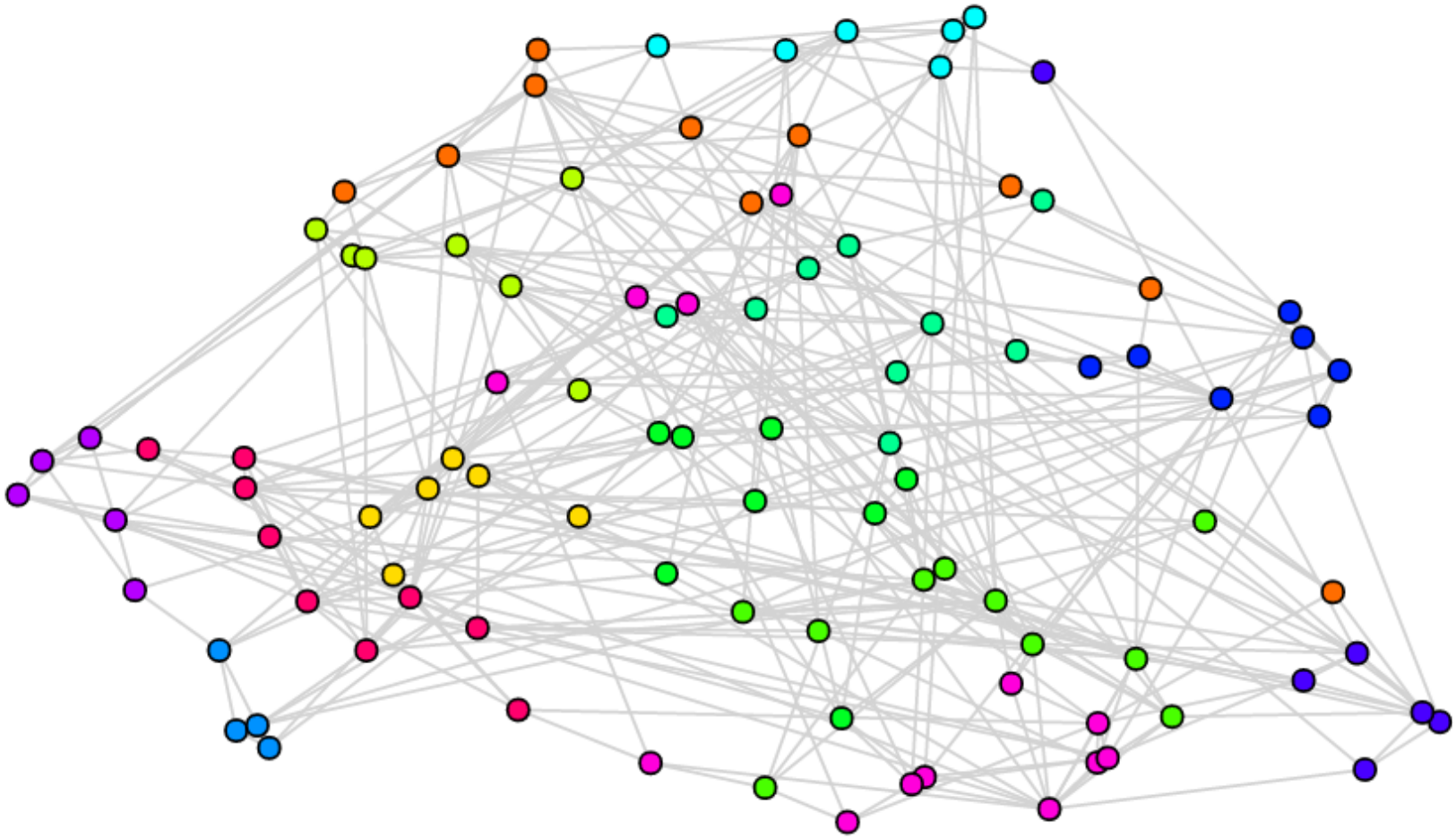
LFR35 - best



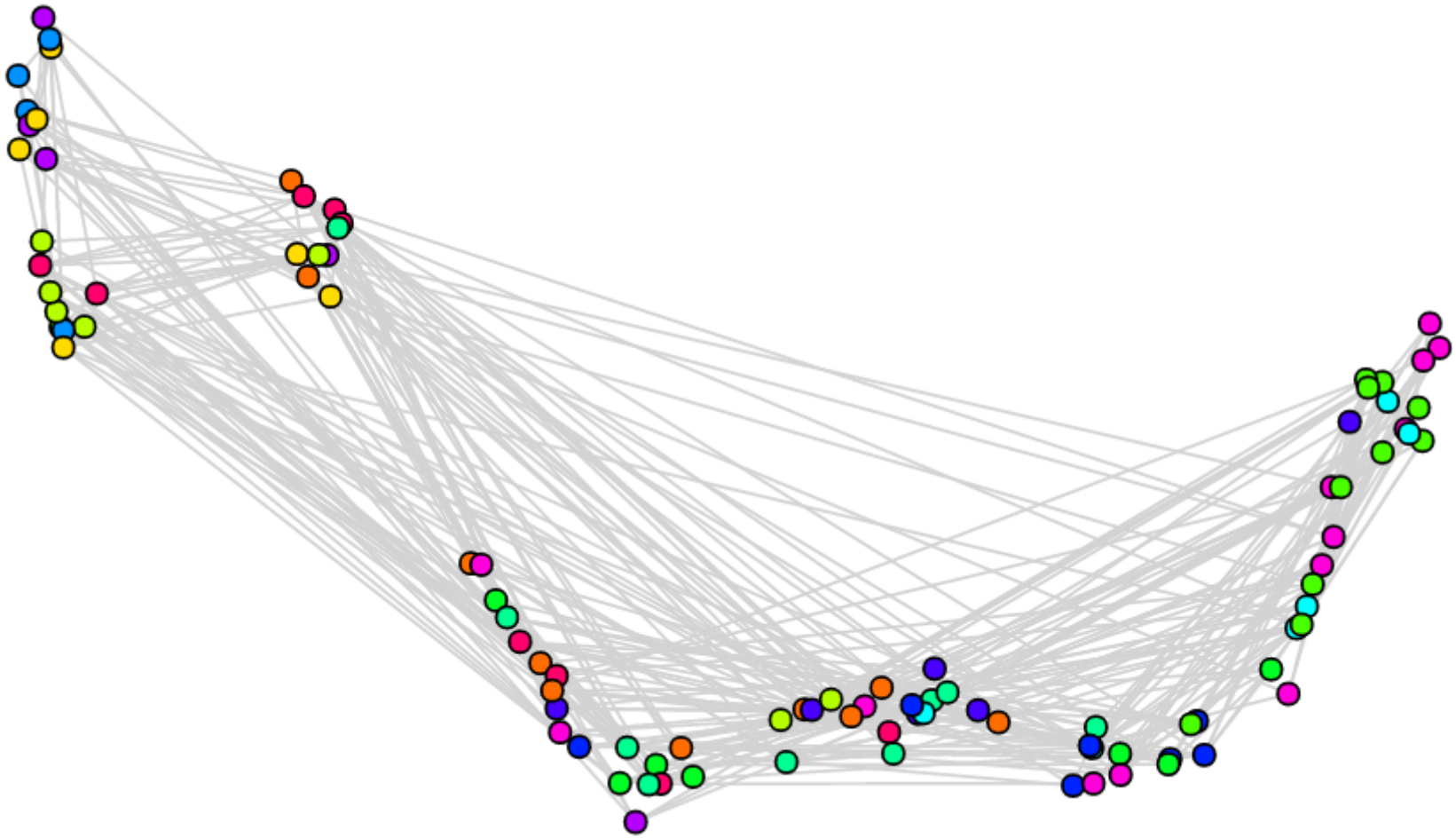
LFR35 - worst



LFR55 - best



LFR55 - worst



Graph Embedding

Notebook #7