A Scalable Unsupervised Framework for Comparing Graph Embeddings

Bogumił Kamiński* Paweł Prałat† François Théberge‡

April 2, 2020

Abstract

Graph embedding is a transformation of vertices of a graph into a set of vectors. A good embedding should capture the graph topology, vertex-to-vertex relationship, and other relevant information about the graph, its subgraphs, and vertices. If these objectives are achieved, an embedding is a meaningful, understandable, and often compressed representations of a network. Finally, vector operations are simpler and faster than comparable operations on graphs.

The main challenge is that one needs to make sure that embeddings well describe the properties of the graphs. In particular, a decision has to be made on the embedding dimensionality which highly impacts the quality of an embedding. As a result, selecting the best embedding is a challenging task and very often requires domain experts.

In the recent paper [1], we propose a "divergence score" that can be assigned to embeddings to help distinguish good ones from bad ones. This general framework provides a tool for an unsupervised graph embedding comparison. In order to achieve it, we needed to generalize the well-known Chung-Lu model to incorporate geometry which is an interesting result in its own right. In order to test our framework, we did a number of experiments with synthetic networks as well as real-world networks, and various embedding algorithms.

The complexity of the original algorithm proposed in [1] was quadratic in the number of vertices. It was enough to show that the proposed method is feasible and has practical potential (proof-of-concept). In this paper, we improve the complexity of the original framework and design a scalable approximation algorithm. We perform some detailed quality and speed benchmarks.

1 Introduction

The study of networks has emerged in diverse disciplines as a means of analyzing complex relational data. Indeed, capturing aspects of a complex system as a graph can bring physical insights and predictive power [2]. Network Geometry is a rapidly developing approach in Network Science [3] which further abstracts the system by modelling the vertices of the network as points in a geometric space. There are many successful examples of this approach that include latent space models [4], and connections between geometry and network clustering and community

*Decision Analysis and Support Unit, SGH Warsaw School of Economics, Warsaw, Poland; e-mail: bogumil.kaminski@sgh.waw.pl

†Department of Mathematics, Ryerson University, Toronto, ON, Canada; e-mail: pralat@ryerson.ca

‡Tutte Institute for Mathematics and Computing, Ottawa, ON, Canada; email: theberge@ieee.org

structure [5, 6]. Very often, these geometric embeddings naturally correspond to physical space, such as when modelling wireless networks or when networks are embedded in some geographic space [7, 8]. See [9] for more details about applying spatial graphs to model complex networks.

Another important application of geometric graphs is in graph embedding. The idea here is that, for a given network, one tries to embed it in a geometric space by assigning coordinates to each vertex such that nearby vertices are more likely to share an edge than those far from each other. In a good embedding most of the network's edges can be predicted from the coordinates of the vertices. For example, in [10] protein interaction networks are embedded in low-dimension Euclidean space. Unfortunately, in the absence of a general-purpose representation for graphs, very often graph embedding requires domain experts to craft features or to use specialized feature selection algorithms. Having said that, there are some graph embedding algorithms that work without any prior or additional information other than graph structure, but these are randomized algorithms that are usually not very stable; that is, the outcome of two applications of the algorithm is often drastically different despite the fact that all the algorithm parameters remain the same.

Consider a graph $G = (V, E)$ on n vertices, and several embeddings of its vertices in some multidimensional spaces (possibly in different dimensions). The main question we try to answer in this paper is: how do we evaluate these embeddings? Which one is the best and should be used? In order to answer these questions, we propose a general framework that assigns the divergence score to each embedding which, in an unsupervised learning fashion, distinguishes good from bad embeddings. In order to benchmark embeddings, we generalize the well-known Chung-Lu random graph model to incorporate geometry. The model is interesting on its own and should be useful for many other problems and tools. In order to test our algorithm, we experiment with synthetic networks as well as real-world networks, and various embedding algorithms.

The paper is structured as follows. In Section 2, we describe our algorithm for comparing graph embeddings, and we illustrate our approach on one simple graph. The Chung-Lu model is generalized in Section 3. Some theoretical results that justify the model can be found in Appendix A. In Section 4, we describe the experiments and present results. In particular, the datasets and embedding algorithms used are outlined respectively in subsections 4.1 and 4.2. Improvements that were required to make an algorithm scalable are presented in Section 5. The results presented in that section for a novel extension of the original algorithm from [1] are the main contribution of this paper. We conclude with a discussion on some future directions in Section 6.

2 General Framework

Suppose that we are given a graph $G = (V, E)$ 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.

2.1 Intuition Behind the Algorithm

What do we expect from a good embedding? As already mentioned, in a good embedding, one should be able to predict most of the network's edges from the coordinates of the vertices. Formally, it is natural to expect that if two vertices, say u and v, are far away from each other (that is, $\mathrm{dist}(\mathcal{E}(u), \mathcal{E}(v))$ is relatively large), then the chance they are adjacent in the graph is smaller compared to another pair of vertices that are close to each other. But, of course, in any real-world network there are some sporadic long edges and some vertices that are close to each other are not adjacent. In other words, we do not want to pay attention to local properties such as existence of particular edges (microscopic point of view) but rather evaluate some global properties such as density of some relatively large subsets of vertices (macroscopic point of view). So, how can we evaluate if the global structure is consistent with our expectations and intuition without considering individual pairs?

The approach we take is as follows. We identify dense parts of the graph by running some good graph clustering algorithm. As we will illustrate in Section 4, the choice of graph clustering algorithm is flexible so long as the vertex set is partitioned into clusters such that there are substantially more edges captured within clusters than between them. The clusters that are found will provide the desired macroscopic point of view of the graph. Note that for this task we only use information about the graph G; in particular, we do not use the embedding $\mathcal{E}$ at all. We then consider the graph G from a different point of view. Using the Geometric Chung-Lu (GCL) model that we introduce in this paper especially for this purpose, based on the degree distribution $\mathbf{w}$ and the embedding $\mathcal{E}$, we compute the expected number of edges within each cluster found earlier, as well as between them. The embedding is scored by computing a divergence score between these expected number of edges, and the actual number of edges present in G. Our approach falls into a general and commonly used method of *statistical inference*, in our case applied to the Geometric Chung-Lu model. With these methods, one fits a generative model of a network to observed network data, and the parameters of the fit tell us about the structure of the network in much the same way that fitting a straight line through a set of data points tells us about their slope.

Finally, let us make a comment that not all embeddings proposed in the literature try to capture edges. Some algorithms indeed try to preserve edges whereas others care about some other structural properties; for example, they might try to map together nodes with similar functions. Because of the applications we personally need to deal with require preserving (global) edge densities, our framework favours embeddings that do a good job from that perspective.

2.2 Algorithm

Given a graph $G = (V, E)$, its degree distribution $\mathbf{w}$ on V, and an embedding $\mathcal{E} : V \to \mathbb{R}^k$ of its vertices in k-dimensional space, we perform the five steps detailed below to 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 via $\mathrm{argmin}_{i \in [m]} \Delta_{\mathcal{E}_i}(G)$ (here and later in the paper, we use $[n]$ to denote the set of natural numbers less than or equal to n; that is, $[n] := \{1, \ldots, n\}$). Note that our algorithm is a general framework and some parts have flexibility. We clearly identify these below.

Step 1: Run some stable *graph* clustering algorithm on G to obtain a partition $\mathbf{C}$ of the vertex set V into ℓ communities $C_1, \ldots, C_\ell$.

Note: In our implementation, we used the ensemble clustering algorithm for graphs (ECG) which is based on the Louvain algorithm and the concept of consensus clustering [11], and is shown to have good stability. We experiment with other algorithms in Section 4.
Note: In some applications, the desired partition can be provided together with a graph (for example, when nodes contain some natural labelling and so some form of a ground-truth is provided).

Step 2: For each $i \in [\ell]$, let c_i be the proportion of edges of G with both endpoints in C_i. Similarly, for each $1 \leq i < j \leq \ell$, let $c_{i,j}$ be the proportion of edges of G with one endpoint in C_i and the other one in C_j. Let

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

be two vectors with a total of $\binom{\ell}{2} + \ell = \binom{\ell+1}{2}$ entries which together sum to one. These *graph vectors* characterize the partition $\mathbf{C}$ from the perspective of the graph G.
Note: The embedding $\mathcal{E}$ does *not* affect the vectors $\bar{\mathbf{c}}$ and $\hat{\mathbf{c}}$. They are calculated purely based on G and the partition $\mathbf{C}$.

Step 3: For a given parameter $\alpha \in \mathbb{R}_+$ and the same vertex partition $\mathbf{C}$, we consider $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$, the GCL Model presented in Section 3. For each $1 \leq i < j \leq \ell$, we compute $b_{i,j}$, the expected proportion of edges of $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$ with one endpoint in C_i and the other one in C_j. Similarly, for each $i \in [\ell]$, let b_i be the expected proportion of edges within C_i. That gives us another two vectors

$$\bar{\mathbf{b}}_{\mathcal{E}}(\alpha) = (b_{1,2}, \ldots, b_{1,\ell}, b_{2,3}, \ldots, b_{2,\ell}, \ldots, b_{\ell-1,\ell}) \qquad \text{and} \qquad \hat{\mathbf{b}}_{\mathcal{E}}(\alpha) = (b_1, \ldots, b_\ell) \tag{2}$$

with a total of $\binom{\ell+1}{2}$ entries which together sum to one. These *model vectors* characterize the partition $\mathbf{C}$ from the perspective of the embedding $\mathcal{E}$.
Note: The structure of graph G does *not* affect the vectors $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$; only its degree distribution $\mathbf{w}$ and embedding $\mathcal{E}$ are used.
Note: We used the Geometric Chung-Lu Model but the framework is flexible. If, for any reason (perhaps there are some restrictions for the maximum edge length; such restrictions are often present in, for example, wireless networks) it makes more sense to use some other model of random geometric graphs, it can be easily implemented here. If the model is too complicated and computing the expected number of edges between two parts is challenging, then it can be approximated easily via simulations.

Step 4: Compute the distances between the two pairs of vectors, that is, between $\bar{\mathbf{c}}$ and $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$, and between $\hat{\mathbf{c}}$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$, in order to measure how well the model $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$ fits the graph G. Let Δ_α be a weighted average of the two distances.
Note: We used the well-known and widely used Jensen–Shannon divergence (JSD) to measure the dissimilarity between two probability distributions. The JSD was originally proposed in [12] and can be viewed as a smoothed version of the Kullback-Leibler divergence. In our implementation, we used simple average, that is,

$$\Delta_\alpha = \frac{1}{2} \cdot \Big(JSD(\bar{\mathbf{c}}, \bar{\mathbf{b}}(\alpha)) + JSD(\hat{\mathbf{c}}, \hat{\mathbf{b}}(\alpha)) \Big) . \tag{3}$$

We decided to independently treat internal and external edges to compensate the fact that there are $\binom{\ell}{2}$ coefficients related to external densities whereas only ℓ ones related to internal ones. Depending on the application at hand, other weighted averages can be used if more weight needs to be put on internal or external edges.

Step 5: Select $\hat{\alpha} = \text{argmin}_\alpha \Delta_\alpha$, and define the *divergence score* for embedding $\mathcal{E}$ on G as: $\Delta_\mathcal{E}(G) = \Delta_{\hat{\alpha}}$.
Note: The parameter α is used to define a distance in the embedding space, as we detail in Section 3. In our implementation we simply checked values of α on a grid between 0 and 10. There are clearly better ways to search the space of possible values of α but, since the algorithm worked very fast on our graphs, we did not optimize this part.

In order to compare several embeddings for the same graph G, we repeat steps 3-5 above and compare the divergence scores (the lower the score, the better). Let us stress again that steps 1-2 are done only once, so we use the same partition of the graph into ℓ communities for each embedding. The code can be accessed at the following GitHub repository[1].

2.3 Illustration

We illustrate our framework on the well-known Zachary's Karate Club graph (see Subsection 4.1 for the description of this and other datasets). Illustrations with other datasets are shown in Appendix C.

The parameter $\alpha \geq 0$ in the GCL model controls the distance used in the embedding space. With $\alpha = 0$, the embedding is *not* taken into account and the classic Chung-Lu model is obtained, so only the degree distribution is accounted for. As α gets larger, long edges in the embedding space are penalized more severely. In the left plot of Figure 1, we show the impact of varying α on the two components of equation (3) which respectively consider pairs of vertices that are *internal* (to some cluster) or *external* (between clusters). Recall that the divergence score for a given embedding is obtained by choosing $\hat{\alpha} = \text{argmin}_\alpha \Delta_\alpha$. In the right plot of Figure 1, we show a 2-dimensional projection of the best embedding as obtained by our framework (with node2vec, 64 dimensions and parameters, p=0.5 and q=1.0). The vertices are coloured according to the two known communities.

We can use the GCL model to generate edges, as with the standard Chung-Lu model. In Figure 2, we generate 3 such graphs using the best embedding shown in Figure 1. The left plot uses $\alpha = 0$, which ignores the embedding and clearly generates too many long edges between the clusters. The center plot uses the optimal value ($\hat{\alpha} = 2.75$ in this case), generating a graph that resembles the true one. The rightmost plot uses the larger value $\alpha = 7$, which penalizes long edges more severely, yielding a graph with less edges between the two communities.

3 Geometric Chung-Lu Model

It is known that classical Erdős-Rényi (binomial) random graphs $G(n, p)$ can be generalized to $G(\mathbf{w})$, the random graph with a given expected degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$. Because of our application, we will define it as a function of a given graph G on n vertices but, in fact, it is only a function of its degree sequence.

Since our goal is to compare different embeddings of the same graph, we will generalize the Chung-Lu model further, including geometry coming from the graph embedding. In such

[1] https://github.com/ftheberge/Comparing_Graph_Embeddings

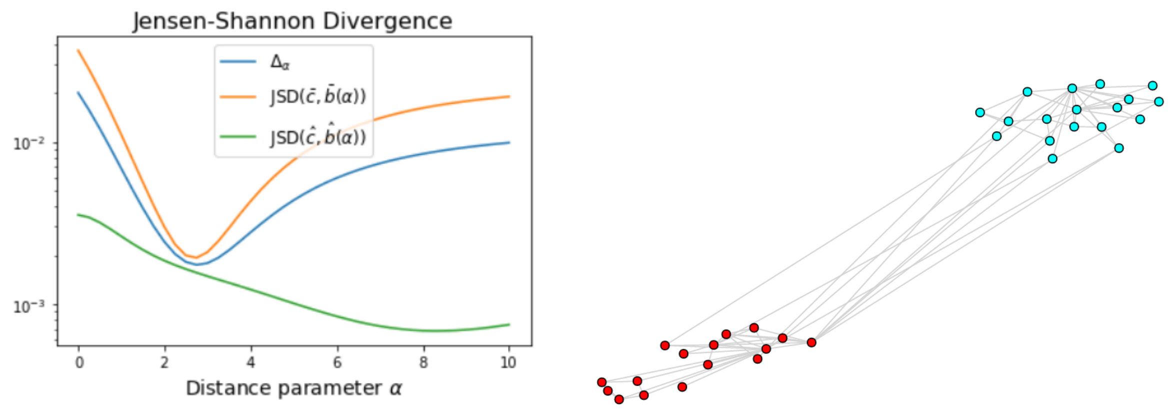

Figure 1: Zachary's Karate Club Graph. We illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). The colors represent the two ground-truth communities.

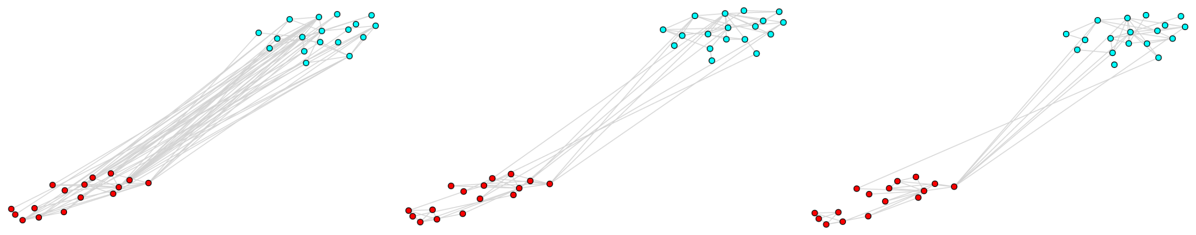

Figure 2: Zachary's Karate Club Graph. We generate random edges following the Geometric Chung-Lu Model with the same expected degree distribution and with the highest scoring embedding. We look at three cases: $\alpha = 0$ which ignores the embedding (left), $\alpha = 7$ which penalizes long edges too severely (right), and the best $\hat{\alpha} = 2.75$ (center).

models, vertices are embedded in some metric space and link formation is influenced by the metric distance between vertices. Such models are called *spatial models* or *geometric graphs*. The main principle of spatial models is that vertices that are metrically close are more likely to link to each other. This is a formal expression of the intuitive notion we have about virtual networks: Web links are likely to point to similar pages, people that share similar interests are more likely to become friends on Facebook, and scientific papers mostly refer to papers on similar topics.

One of the first geometric random graph models in which the probability of an edge is discounted with the distance was proposed by Waxman [13]. The Waxman model has been generalized to the class of random geometric graphs with other distance discounting functions; see, for example, [14]. In another important work in this area, Spatially Embedded Random Networks [15] are introduced with a goal to gain an intuition for the effects of spatial embedding on network structure. Currently, there are many interesting random geometric models of complex networks known in the literature, including hyperbolic random graphs [16] and Spatial Preferential Attachment model [17] introduced by the second author of this paper and his coauthors. We direct the reader to [9] for more details about applying spatial graphs to model complex networks.

Let us explicitly mention about the following two models that are similar to the one we propose and use in our paper. The first one is called the spatial inhomogeneous random graph on $\mathbb{Z}^d$ [18] and the second one is called the geometric inhomogeneous random graph (GIRG) [19]. They are similar to our model as they create edges with probability that is a function of a distance between the vertices (however, the probability functions are all different). They also have a parameter that describes the long-range nature of the model. The main difference is with the position of the vertices: in [18], vertices form an infinite d-dimensional grid wheres the vertex set in [19] consists of n vertices that are drawn uniformly at random from d-dimensional torus. In our model, since we apply it to evaluate embeddings with fixed positions of vertices, we need a model in which positions of the vertices are given and general.

3.1 Original Model

Let $G = (V, E)$ be a graph, where $V = \{v_1, \ldots, v_n\}$ are the vertices, the edges E are multisets of V of cardinality 2 (loops are allowed), and $\deg_G(v)$ is the degree of v in G (with a loop at v contributing 2 to the degree of v). We define $\mathcal{G}(G)$ to be the probability distribution on graphs on the vertex set V following the well-known Chung-Lu model [20, 21, 22, 23]. In this model, each set $e = \{v_i, v_j\}$, $v_i, v_j \in V$, is independently sampled as an edge with probability given by:

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

Let us mention one technical assumption. It might happen that $P(v_i, v_j)$ is greater than one and so it should really be regarded as the expected number of edges between i and j; for example, as suggested in [24] (see also a book of Newman [2]), one can introduce a Poisson-distributed number of edges with mean $P(v_i, v_j)$ between each pair of vertices i, j. However, since typically the maximum degree D satisfies $D^2 \leq 2|E|$ it rarely creates a problem and so we may assume that $P(v_i, v_j) \leq 1$ for all pairs.

As already mentioned, this model is a function of the degree sequence of G. One desired property of this random model is that it yields a distribution that preserves the expected degree for each vertex, namely: for any $i \in [n]$,

$$\begin{aligned} \mathbb{E}_{G' \sim \mathcal{G}(G)}[\deg_{G'}(v_i)] &= \sum_{j \in [n] \setminus \{i\}} \frac{\deg_G(v_i)\deg_G(v_j)}{2|E|} + 2 \cdot \frac{\deg_G^2(v_i)}{4|E|} \\ &= \frac{\deg_G(v_i)}{2|E|} \sum_{j \in [n]} \deg_G(v_j) \quad = \quad \deg_G(v_i). \end{aligned}$$

3.2 Geometric Model

In the Geometric Chung-Lu model we are not only given the expected degree distribution of a graph G

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

but also an embedding of vertices of G in some k-dimensional space, function $\mathcal{E} : V \to \mathbb{R}^k$. In particular, for each pair of vertices, v_i, v_j, we know the distance between them:

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

It is desired that the probability that vertices v_i and v_j are adjacent to be a function of $d_{i,j}$, that is, to be proportional to $g(d_{i,j})$ for some function g. The function g should be a decreasing function as long edges should occur less frequently than short ones. There are many natural choices such as $g(d) = d^{-\beta}$ for some $\beta \in [0, \infty)$ or $g(d) = \exp(-\gamma d)$ for some $\gamma \in [0, \infty)$. We use the following, normalized function $g : [0, \infty) \to [0, 1]$: for a fixed $\alpha \in [0, \infty)$, let

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

where

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

are the minimum, and respectively the maximum, distance between vertices in embedding $\mathcal{E}$. One convenient and desired property of this function is that it is invariant with respect to an affine transformation of the distance measure. Clearly, $g(d_{\text{min}}) = 1$ and $g(d_{\text{max}}) = 0$; in the computations, we can use clipping to force $g(d_{\text{min}}) < 1$ and/or $g(d_{\text{max}}) > 0$ if required. Let us also note that if $\alpha = 0$ (that is, $g(d) = 1$ for any $d \in [0, \infty)$ with $g(d_{\text{max}}) = 0^0 = 1$), then we recover the original Chung-Lu model as the pairwise distances are neglected. Moreover, the larger parameter α is, the larger the aversion to long edges is. Since this family of functions (for various values of the parameter α) captures a wide spectrum of behaviours, it should be enough to concentrate on this choice but one can easily experiment with other functions. So, for now we may assume that the only parameter of the model is $\alpha \in [0, \infty)$.

The *Geometric Chung-Lu* (GCL) model is the random graph $G(\mathbf{w}, \mathcal{E}, \alpha)$ on the vertex set $V = \{v_1, \ldots, v_n\}$ in which each pair of vertices v_i, v_j, independently of other pairs, forms an edge with probability $p_{i,j}$, where

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

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; that is, for all $i \in [n]$

$$w_i = \sum_{j \in [n], j \neq i} p_{i,j} = x_i \sum_{j \in [n], j \neq i} x_j g(d_{i,j}).$$

Additionally, we set $p_{i,i} = 0$ for $i \in [n]$. Indeed, we prove in Appendix A that there exists the unique selection of weights, provided that the maximum degree of 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. Finally, let us mention that in Appendix A it is assumed that $g(d_{i,j}) > 0$ for all pairs i, j. In our case, $g(d_{i,j}) = 0$ for a pair of vertices that are at the maximum distance. It causes no problems in practice but, as mentioned earlier, one can easily scale the outcome of function $g(\cdot)$ to move away from zero without affecting the divergence score in any non-negligible way.

It is not clear how to find weights explicitly but they can be easily (and efficiently) approximated numerically to any desired precision. In Appendix A, we prove that, if the solution exists, which is easy to check, then the set of right hand sides of the equations, considered as a function from $\mathbb{R}^n$ to $\mathbb{R}^n$, is a local diffeomorphism everywhere in its domain. As a result, standard gradient root-finding algorithms should be quite effective in finding the desired weights. We discuss one simple numerical approximation procedure in the next subsection.

3.3 Numerical Approximation

Let us start with an arbitrary vector $\mathbf{t}^0 = (t_1^0, \ldots, t_n^0)$ (say, $\mathbf{t}^0 = (1, \ldots, 1)$) that we will carefully tune by iteratively constructing a sequence $(\mathbf{t}^s)_{s \geq 0}$. Suppose that we are given a vector $\mathbf{t}^s = (t_1^s, \ldots, t_n^s)$. If we introduce an edge between v_i and v_j with probability

$$p_{i,j}^s = t_i^s t_j^s g(d_{i,j}), \tag{4}$$

then the expected degree of v_i would be

$$s_i^s = \sum_j p_{i,j}^s = t_i^s \sum_j t_j^s g(d_{i,j}). \tag{5}$$

Note that, for a given vertex v_i, it easy to adjust the weights so that s_i^s matches w_i, the desired expected degree of v_i; indeed, one can replace t_i^s with $t_i^s(w_i/s_i^s)$. However, unfortunately, it will also affect other values of $\mathbf{s}^s$ and vice versa, changes in other parts of $\mathbf{t}$ affect s_i^s too. Hence, instead of doing it, each vertex should take a small step into the right direction and this process should quickly converge to the desired state: s_i^s being very close to w_i for all i. Let us note that, for our application, we do not need to get close to the desired expected degree sequence. The divergence score we defined earlier is not too sensitive. Fix some small constants $\varepsilon, \delta > 0$. For example, in our experiments we used $\varepsilon = 0.1$ and $\delta = 0.001$. For each i, $1 \leq i \leq n$, we define

$$t_i^{s+1} = (1 - \varepsilon) t_i^s + \varepsilon t_i^s (w_i / s_i^s) = t_i^s + \varepsilon t_i^s (w_i / s_i^s - 1). \tag{6}$$

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

4 Experiments

4.1 Datasets

In order to test our algorithm, we benchmark it against synthetic graphs with communities as well as some real-world graphs with known community structures. This is a natural and often used approach (see, for example, [25]). In all examples we colour the vertices with respect to the known, ground-truth communities, but in the algorithm, we use the partition obtained via a graph clustering algorithm, as the algorithm is unsupervised.

4.1.1 LFR

As common in the literature, we analyze how the algorithm performs on artificially constructed networks with communities. A popular example is the LFR benchmark by Lancichinetti, Fortunato, and Radicchi [26] that generates synthetic graphs that resemble real world graphs. In this benchmark, the size of each community is drawn from a power-law distribution, as is the degree of each vertex.

The LFR model has a number of parameters. The most important one is the mixing parameter μ that controls the fraction of *external* edges (that is, edges between communities). Essentially this can be viewed as the amount of noise in the graph. In one extreme case, if $\mu = 0$, then all the edges are within communities. On the other hand, if $\mu = 1$, then all edges are between different communities. Other parameters control the number of vertices (n), the negative exponent of the power-law degree distribution (γ_1), the negative exponent of the distribution of community sizes (γ_2), and the average (d) and the maximum degree ($d_{\max}$).

We generated small LFR graphs with $n = 100$ for visualization purposes. For the mixing parameter μ, we used $\mu = 0.15$ (strong communities, low noise), $\mu = 0.35$ (intermediate case), and $\mu = 0.55$ (noisy graph, weaker communities). For the other parameters, we used, $\gamma_1 = 2$, $\gamma_2 = 1$, $d = 8$, and $d_{\max} = 20$.

4.1.2 ABCD

Unfortunately, the standard method for generating artificial networks, the **LFR** graph generator introduced above, has some scalability limitations and it is challenging to analyze it theoretically. Moreover, the mixing parameter μ, the main parameter of the model guiding the strength of the communities, has a non-obvious interpretation and so can lead to unnaturally-defined networks.

In [27], the authors of this paper provided an alternative random graph model with community structure and power-law distribution for both degrees and community sizes, the **A**rtificial **B**enchmark for **C**ommunity **D**etection (**ABCD** graph). The new model solves the three issues identified above and more. We use it in this paper to generate large ABCD graph on $n = 10{,}000$ and $n = 100{,}000$ vertices to test the approximation algorithm on some larger instances of graphs—see Section 5. For the counterpart of the mixing parameter for LFR, we used $\xi = 0.2$. For the other parameters we used $\gamma_1 = 3$, $\gamma_2 = 2$, average degree $d = 20$ and maximum degree $d_{\max} = 100$ for $n = 10{,}000$, and $d = 25$, $d_{\max} = 500$ for $n = 100{,}000$.

4.1.3 Zachary's Karate Club

This well-known social network consists of 34 members of a karate club and 78 pairwise links observed over a period of three years. During the famous study of Zachary [28], a political conflict arose between the club president and the instructor which caused the club to split into two parts (the communities), each with half of the members. Zachary recorded a network of friendships among members of the club shortly before the fission.

4.1.4 College Football

This graph represents the schedule of United States football games between Division IA colleges during the regular season in Fall 2000 [29]. This is another well-studied, more complex, real-world network with known community structures. The data consists of 115 teams (vertices) and 613 games (edges). The teams are divided into conferences containing 8–12 teams each. In general, games are more frequent between members of the same conference than between members of different conferences, with teams playing an average of about seven intra-conference games and four inter-conference games in the 2000 season. There are a few exceptions to this rule, as detailed in [25]: one of the conferences is really a group of independent teams, one conference is really broken into two groups, and 3 other teams play mainly against teams from other conferences. We refer to those as *outlying* vertices, which we represent with a distinctive triangular shape.

4.1.5 Email-Eu-core Network

This network was generated using email data from a large European research institution and is available as one of the SNAP Datasets [30]. There is a directed edge (u, v) in the network if person u sent person v at least one email. The e-mails only represent communication between institution members (the core), and the dataset does not contain incoming messages from or outgoing messages to the rest of the world. The dataset also contains community memberships of the vertices: each individual belongs to exactly one of 42 departments at the research institute.

There are 1005 vertices and 25571 edges in the original directed graph. After making it undirected and keeping only the largest connected component, we ended up with a graph on 986 vertices and 16064 edges. Most of the communities are very weak; in fact, only 1 community qualifies as a *weak community* as defined in equation (9.2) in [31]; that is, a community for which the ratio between the total internal degree and the total degree is above 0.5. In our study, we looked at the communities for which this ratio was the highest.

4.2 The Graph Embeddings

We considered three popular graph embedding algorithms in our examples: *node2vec*, VERSE and LINE. Recall that our goal is to experiment with our framework and not to do extensive comparison of graph embeddings. Comparisons are done in the original dimension of the embedding. For visualization purposes, we project the embeddings into 2-dimensional space.

4.2.1 node2vec

node2vec [32] is an algorithm for scalable feature learning in networks. Intuitively, the goal is to find feature representations that maximize the likelihood of preserving network neighbourhoods of vertices in a d-dimensional feature space. By choosing an appropriate notion of a neighbourhood, node2vec can learn representations that organize vertices based on their network roles and/or communities they belong to. It is achieved by developing a family of biased random walks, which efficiently explore diverse neighbourhoods of a given vertex.

There are two main parameters in this algorithm. The "return parameter" p controls the likelihood of immediately revisiting a vertex in the random walk. Setting it to a high value ensures that we are less likely to sample an already-visited vertex in the following two steps (unless, of course, the next vertex in the walk had no other neighbours). The "in-out parameter" q allows the search to differentiate between "inward" and "outward" vertices. Loosely speaking, this parameter guides the neighbourhood sampling strategy which allows us to smoothly interpolate between *breadth-first-search* (BFS) and *depth-first search* (DFS). For *node2vec*, we fixed parameter $q = 1$ and we varied parameter $0.5 \leq p \leq 2$. We considered embedding dimensions $2 \leq D \leq 128$. For the other parameters we used the default values.

Finally, let us mention that *node2vec* is, in fact, a semi-supervised algorithm in which the best set of parameters can be done with a small amount of labelled data. However, as mentioned above, we simply considered a grid of possible parameters and compared several different embeddings of the same graph.

4.2.2 VERSE

VERtex Similarity Embeddings (VERSE) [33] is a simple, versatile, and memory-efficient method that derives graph embeddings explicitly calibrated to preserve the distributions of a selected vertex-to-vertex similarity measure. It is a general framework that learns *any* similarity measures among vertices via training a simple, yet expressive, single-layer neural network. This includes popular similarity measures such as Personalized PageRank (PPR), SimRank, and adjacency similarity. We used the default recommended parameters for this algorithm and we considered embedding dimensions $2 \leq D \leq 128$.

Let us also mention about the connection between similarity measures, graph clustering, and embedding that has been studied in [34, 35] where, in particular, the authors discuss the classical Schoenberg embedding [36, 37].

4.2.3 LINE

Large-scale Information Network Embedding (LINE) [38] is an efficient method for vertex embedding that uses an approximate factorization of the adjacency matrix, trying to preserve first as well as second order proximities. We used the default recommended parameters for this algorithm, but we varied the number of threads used from 1 (default) to 10. We considered embedding dimensions $2 \leq D \leq 128$.

4.3 Visualization

Dimension reduction seeks to produce a low dimensional representation of high dimensional data that preserves relevant structure. As a result, it is an important problem in data science as a potential pre-processing step for many machine learning algorithms. Here, we use it for visualization of tested graph embeddings that are done in high dimensions. We used UMAP (Uniform Manifold Approximation and Projection) [39], a novel manifold learning technique for dimension reduction. UMAP is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. It provides a practical scalable algorithm that applies to real world datasets.

4.4 Results

4.4.1 Artificial Graphs

For the three LFR graphs described earlier, we generated 5 embeddings for every choice of parameter(s) and for each algorithm considered, for a total of 315 embeddings for each graph. In Figure 3, we plot a 2-dimensional projection of the best and worst results as identified by our divergence score. The colours correspond to the ground-truth communities generated by the

LFR benchmark. For the best embeddings, we clearly see that the ground-truth communities are grouped together, even (but to a lesser extent) for the noisy graph with $\mu = 0.55$. This is not the case for the worst embeddings presented in the right column of Figure 3.

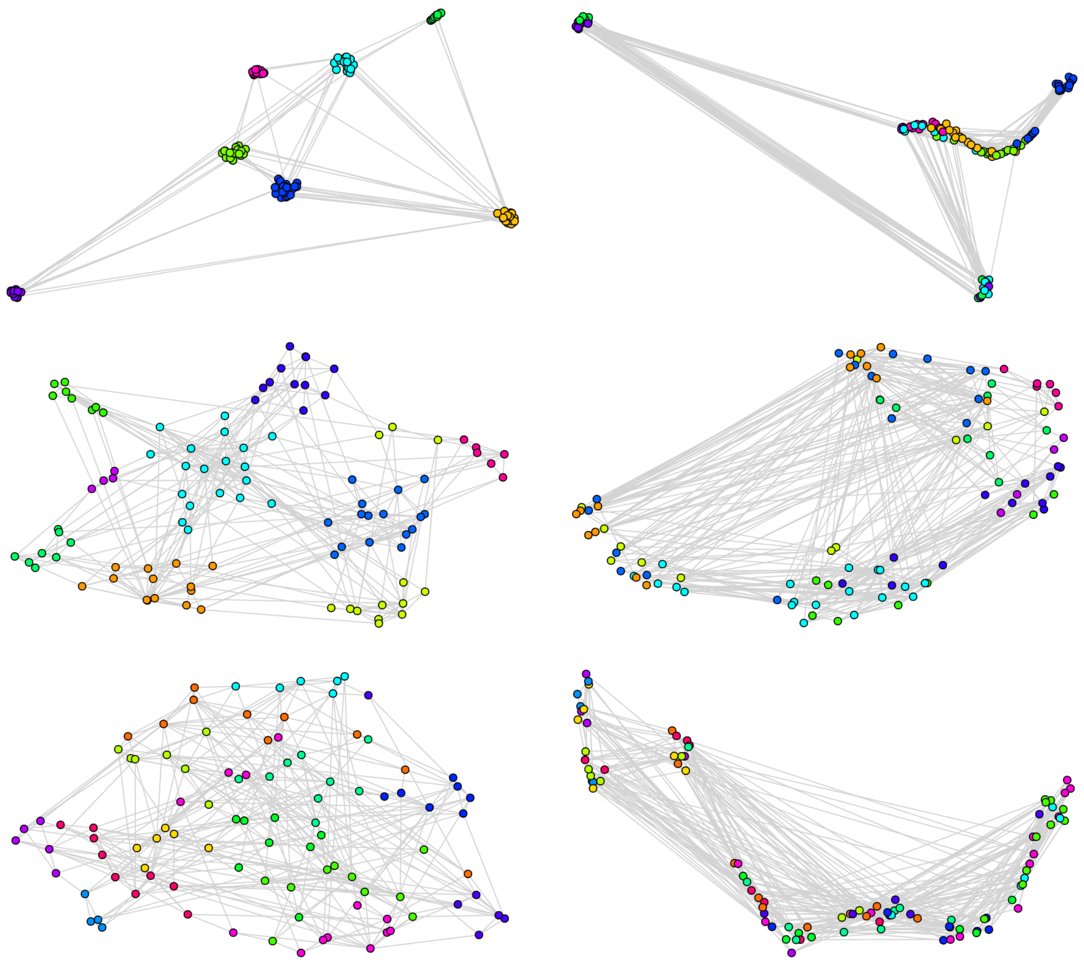

Figure 3: Best (left column) and worst (right column) scoring embeddings for 3 different LFR graphs. In the top row, we show the results for the graph with 7 strong communities ($\mu = 0.15$). In the middle row, we show the results for the graph with $\mu = 0.35$ and 9 communities. Finally in the bottom row, we show the results for the noisy graph ($\mu = 0.55$) with 13 communities.

4.4.2 Zachary's Karate Club Network

We already considered this dataset in Section 2. In Figure 4, we show the best and worst scoring embeddings as determined by our framework. The colours correspond to the 2 real communities. The left plot is clearly a better choice.

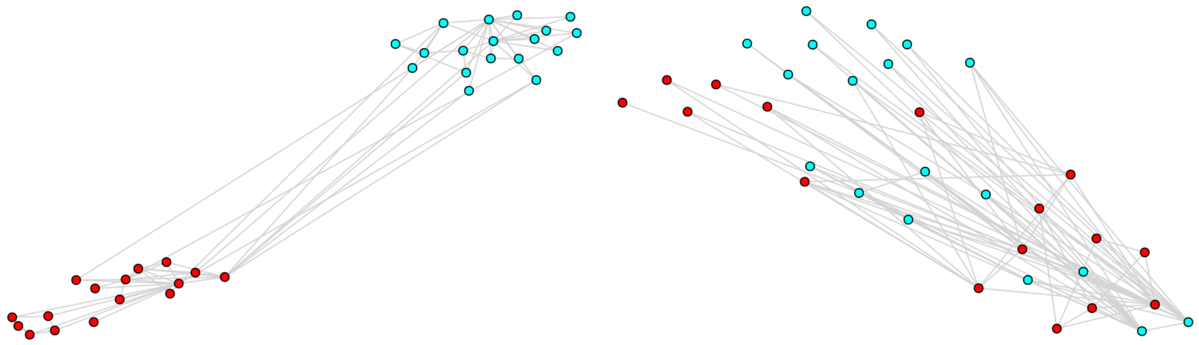

Figure 4: Zachary's Karate Club. We show the best (left) and the worst (right) embedding according to our divergence score

.

4.4.3 The College Football Graph

This graph consists of 115 teams spread between 12 conferences. In Figure 5, we show the best and worst scoring embeddings. The colours of vertices correspond to the conferences, white triangular shaped vertices correspond to outlying vertices as observed earlier. The communities are very clear in the left plot.

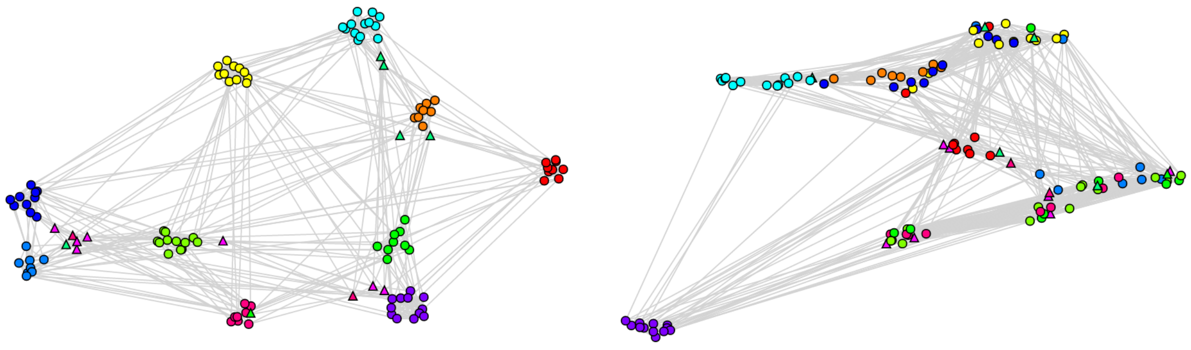

Figure 5: The College Football Graph. We show the best (left) and the worst (right) scoring embedding.

For each choice of embedding model, parameters and dimension, we obtained 5 different embeddings. In Figure 6, we summarize those results by showing the mean and standard deviation of the divergence score $\Delta_{\mathcal{E}}(G)$. This allows us to view some overall properties: for example, embedding in 2-dimensions is not enough in this case; we obtained the best result with LINE with

32 dimensions or more. Recall that we are not performing an overall study of graph embedding algorithms, and we are not looking for a winning model. In fact, both VERSE and *node2vec* gave the best results for other graphs, which indicates the need for a comparison framework.

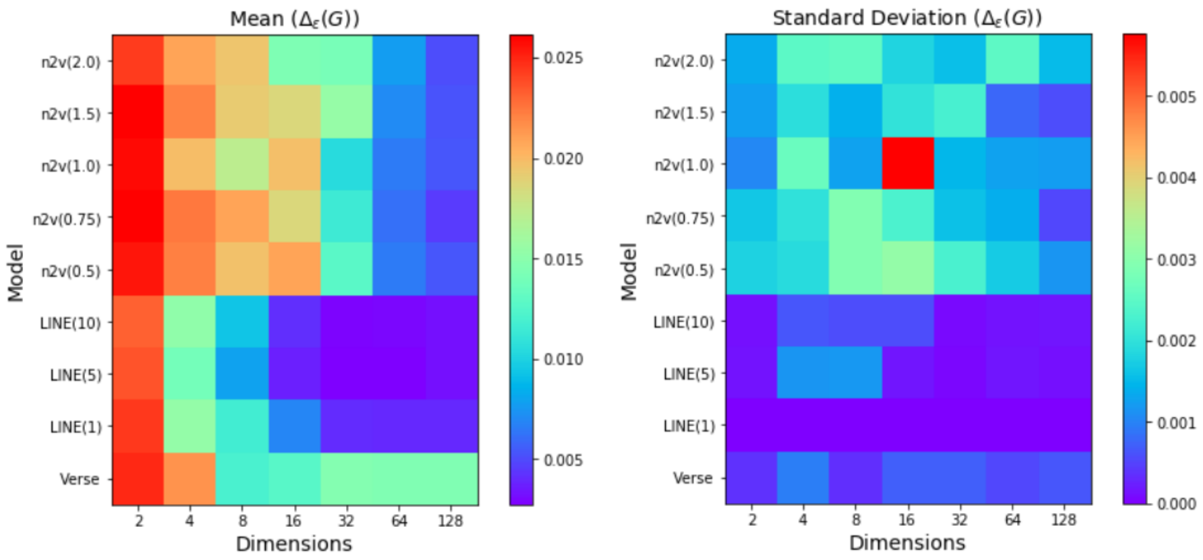

Figure 6: For the College Football Graph, for each choice of algorithm, parameters and dimension, we generated 5 embeddings. We show the mean and standard deviation of the divergence score $\Delta_{\mathcal{E}}(G)$ for each of those.

4.4.4 The email-Eu-core Graph

In Figure 7, we show the highest scoring embedding we obtained for the email graph described earlier. Since most communities are very weak, we highlight only the 3 strongest communities for which the ratios of internal to total degree are respectively 0.54, 0.42 and 0.39. These communities are clearly separated.

4.4.5 Graph Clustering

An important part of our algorithm consists of building a reasonable partition of the vertices, over which we build the representative vectors (1) and (2). The ECG algorithm was shown to generally yield good and stable clusters [40], and we used it throughout. We re-ran all embeddings on the College Football graph using respectively the Louvain [41] and InfoMap [42] clustering algorithms, and in each case, we produced a ranking of the embeddings with respect to our method. We compared those rankings as well as the ranking obtained with the ECG clustering algorithm by computing the Kendall-tau correlation between those. The results, which are summarized in Table 1, show very high correlation.

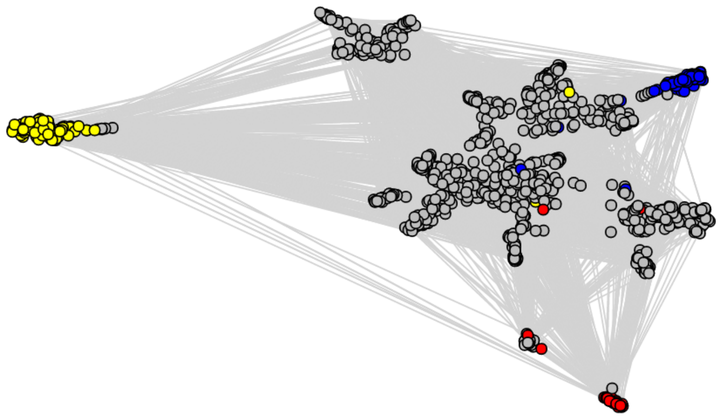

Figure 7: The best scoring embedding we obtained for the email-Eu-core graph with the 3 strongest communities respectively shown in yellow, red and blue.

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.

5 Complexity — Scalable Algorithm

The original algorithm proposed in [1] has a running time that is quadratic as a function of the number of vertices. It was enough to experiment with graphs on a few thousands of vertices to show that the proposed method is feasible and has practical potential (the so-called proof-of-concept). In this section, we improve the complexity and design a scalable algorithm that efficiently evaluates graph embeddings even on millions of vertices.

The main bottleneck of the original algorithm is the process of tuning n weights $x_i \in \mathbb{R}_+$ ($i \in [n]$) in the Geometric Chung-Lu model (Step 3 of the algorithm). This part requires $\Theta(n^2)$ steps and so it is not feasible for large graphs. The other components are much faster with the graph clustering algorithm (Step 1 of the algorithm) being the next computationally intensive part, typically requiring $O(n \ln n)$ steps. We modify our algorithm slightly to obtain a scalable approximation algorithm that can be efficiently run on large networks. Its running time is $O(n \ln n)$ which is practical. Indeed, let us point out that graph embedding algorithms have their own non-trivial complexity and so our benchmark framework is certainly not a bottleneck of the whole process anymore.

Recall that in Part 3 of the algorithm, for a given parameter $\alpha \in \mathbb{R}_+$ and vertex partition $\mathbf{C}$, we need to compute the expected proportion of edges of $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$ that are present within partition parts and between them, vectors $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$ defined in (2). The main idea behind our approximation algorithm is quite simple. Our goal is to group together vertices from the same part of $\mathbf{C}$ that are close to each other in the embedded space. Once such refinement of partition $\mathbf{C}$ is generated, we simply replace each group by the corresponding auxiliary vertex that is placed in the (appropriately weighted) center of mass of the group it is associated with. Such auxiliary vertices will be called **landmarks**. Finally, vectors $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$ will be approximated by vectors $\bar{\mathbf{a}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{a}}_{\mathcal{E}}(\alpha)$ in the corresponding auxiliary graph of landmarks. Since we aim for a fast algorithm, the numer of landmarks should be close to $n' = \sqrt{n}$ so that the process of tuning weights can be done in $O(n'^2) = O(n)$ time.

The process of selecting landmarks is discussed in the next subsection but let us mention about one more modification that needs to be done. Our initial Geometric Chung-Lu model produces simple graphs. On the other hand, after merging vertices from one group into the corresponding landmark, we need to control the expected number of edges between these vertices. Hence, we need to generalize our model to include loops which we discuss in the following subsection before we move to the quality and speed comparison.

Generating Landmarks

We start with a partition $\mathbf{C}$ of the vertex set V into ℓ communities $C_1, \ldots, C_\ell$. The number of communities is typically relatively small. In what we write below, our mild assumption is that $\ell < \sqrt{n}$; otherwise, one may simply use the original algorithm or increase the number of landmarks (alternatively, one may insist that the number of initial communities produced by graph clustering algorithm is small). For each part C_i ($i \in [\ell]$) we compute the **weighted center of mass** p_i and the **weighted sum of squared errors** (**SSE**) e_i, that is,

$$p_i := \frac{\sum_{j \in C_i} w_j \ \mathcal{E}(v_j)}{\sum_{j \in C_i} w_j} \qquad \text{and} \qquad e_i = \sum_{j \in C_i} w_j \ \mathrm{dist}\big(p_i, \mathcal{E}(v_j)\big)^2.$$

(Recall that w_j is the degree of vertex v_j and $\mathcal{E}(v_j)$ is its position in the embedded space R^k.) The weighted sum of squared errors is a natural measure of variation within a cluster.

We will refine the partition $\mathbf{C}$ by repeatedly splitting some parts of it with the goal to reach precisely $\sqrt{n}$ parts. However, before we explain which parts will be split, let us concentrate on splitting a given part C_i. The goal is to partition C_i with SSE equal to e_i into two parts with the corresponding SSEs equal to e_i^1 and e_i^2 in such a way that $\max\{e_i^1, e_i^2\}$ is as small as possible. Finding the best partition is difficult and computationally expensive. However, this can be efficiently well approximated by finding the first principal component in the well-known **weighted Principal Component Analysis** (**PCA**). This transformation is defined in such a way that the first principal component has the largest possible weighted variance (that is, accounts for as much of the weighted variability as possible). After projecting all the points from C_i onto this component, we get a total order of these points and one can quickly check which of the natural $|C_i| - 1$ partitions minimizes $\max\{e_i^1, e_i^2\}$. The original part C_i is then replaced with two parts, C_i^1 and C_i^2, with the corresponding centers of mass and SSEs. See Figure 8 for an illustration of this process.

Splitting C_i into two parts so as to minimize $\max\{e_i^1, e_i^2\}$ can be achieved in $O(|C_i|)$ steps using the bisection search over a projection of data onto the first principal component. Indeed, this is doable because of the following: (1) finding the first principal component and calculating the projection onto it has linear cost, (2) finding the median over some range has a linear cost, (3) when a splitting hyperplane is moved, and as a consequence points between C_i^1 and C_i^2 are

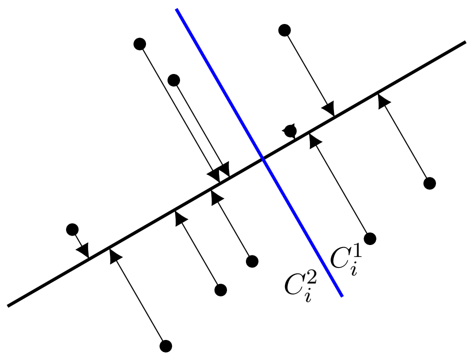

Figure 8: Splitting C_i using SSE and the first principal component. Dots represent original points, thick black line represents the first principal component, and blue line represents the hyperplane orthogonal to the first principal component. It provides the desired split of C_i into C_i^1 and C_i^2.

moved, then e_i^1 and e_i^2 can be updated using an online algorithm that also has a linear cost, and (4) the number of potentially moved points in the bisection search is halved in each step.[2] As a result, since we will be recursively applying splitting until reaching $\sqrt{n}$ parts, similarly as in the case of well-known Quicksort sorting algorithm, the expected total running time of this part of the algorithm is $O(n \ln n)$.

Now, we are ready to describe the strategy for selecting parts for splitting. First of all, let us mention that it is not desired to replace the whole original part by one landmark as it may introduce large error. Indeed, the intuition is that replacing many vertices with a landmark might affect the expected number of edges between them but the expected number of edges between vertices that belong to different landmarks (that are often far away from each other) is not affected too much. As a result, in our implementation we insist on splitting each original part into s smaller parts even if the original SSE is small. (s is a parameter of the model that we will discuss soon.) After this initial phase we start splitting parts in a greedy fashion, each time selecting a part that has the largest SSE. The process stops once $n' = \sqrt{n}$ parts are generated.

Let us now briefly discuss the influence of the parameter s. In a typical scenario, even if s is small, each cluster is split many times in the second part of the process where we greedily split clusters with large SSE. In such situations, the value of the parameter s actually does not matter and this is what we observed in our experiments. However, it is theoretically possible that in some rare cases this natural splitting might not happen. As a result, in the implementation we provided, we allow the user to tune parameter s to cover such rare instances.

Now, let us come back to the algorithm. As already mentioned, each part C_i is replaced by its landmark u_i. The position of landmark u_i in the embedded space $\mathbb{R}^k$ coincides with the weighted center of mass of its part, that is, $\mathcal{E}(u_i) = p_i$. Finally, the expected degree of landmark u_i (that we denote as w_i' in order to distinguish it from w_i, the expected degree of vertex w_i) is the sum of the expected degrees of the associated vertices in the original model, that is, $w_i' := \sum_{j \in C_i} w_j$.

[2]See `split_cluster_rss` function in the reference implementation. In the code we make a significant use of the fact, that the Julia language provides an efficient implementation of views into arrays which allowed us to keep the number of required memory allocations made in the code small.

Note: We experimented with a number of different strategies for splitting, other than minimizing the maximum SSE, such as balancing sizes of all clusters and balancing diameters of all clusters. Once the objective function is fixed, the algorithm may greedily select the worst cluster (from a given perspective) and then split it appropriately (again, to minimize the objective function). The results were comparable but the best outcome turned out to be with the approach we described above.

Including Loops in the Geometric Chung-Lu Model

In order to approximate vectors $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$ from the original model on n vertices, we will use the auxiliary model on $n' = \sqrt{n}$ landmarks. Each landmark u_i is located at $p_i \in \mathbb{R}^k$ (the weighted center of mass of the associated vertices) and has expected degree w_i (the sum of expected degrees of the associated vertices). One can find the pairwise distances between landmarks, and apply the original model $G(\mathbf{w}, \mathcal{E}, \alpha)$ for landmarks to compute the expected number of edges between and within parts, vectors $\bar{\mathbf{a}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{a}}_{\mathcal{E}}(\alpha)$, as an approximation of the original vectors $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$ and $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$. It is expected that $\bar{\mathbf{a}}_{\mathcal{E}}(\alpha)$ approximates well $\bar{\mathbf{b}}_{\mathcal{E}}(\alpha)$ but, since many vertices ($\sqrt{n}$ on average) are reduced to one landmark, the number of internal edges might be affected, that is, $\hat{\mathbf{a}}_{\mathcal{E}}(\alpha)$ might not be very close to $\hat{\mathbf{b}}_{\mathcal{E}}(\alpha)$.

We partially address this issue by insisting that each original part is split into a number of landmarks. In order to achieve even better approximation we introduce loops in our Geometric Chung-Lu Model. This generalization is straightforward. The *Geometric Chung-Lu* (GCL) model is the random graph $H(\mathbf{w}, \mathcal{E}, \alpha)$ on the set of landmarks $V = \{u_1, \ldots, u_{n'}\}$ in which each pair of landmarks u_i, u_j, independently of other pairs, forms an edge with probability $p_{i,j}$, where

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

for some carefully tuned weights $x_i \in \mathbb{R}_+$. Additionally, for $i \in [n']$, the probability of creating a self loop around landmark u_i is equal to

$$p_{i,i} = x_i^2 g(d_{i,i}), \qquad \text{where} \qquad d_{i,i} = \sqrt{\frac{e_i}{\sum_{j \in C_i} w_j}}.$$

Note that the "distance" $d_{i,i}$ from landmark u_i to itself is an approximation of the unobserved weighted average distance $d_{a,b}$ over all pairs of vertices a and b associated with u_i. The weights are selected such that the expected degree of landmark u_i is w'_i; that is, for all $i \in [n']$

$$w'_i = \sum_{j \in [n']} p_{i,j} = x_i \sum_{j \in [n']} x_j g(d_{i,j}).$$

In Appendix B, we revisit the proof of the uniqueness of weights. Finally, let us mention that, as in the case of the original model, standard root-finding algorithms can be used to efficiently find the desired weights.

Quality and Speed Comparison

We start our experiments with revisiting the College Football graph and testing the same set of embeddings. For each embedding, we compared the original divergence score computed for

$n = 115$ vertices with the approximated counterpart computed for $n' = 36$ landmarks. (The value of 36 was selected rather arbitrarily; the graph is tiny so any number of landmarks seems reasonable for this illustration purpose.) Each of the 12 clusters were forced to split once before greedy strategy was applied. The graph presented in Figure 9 shows very high correlation between the two measures which indicates that the approximation algorithm preforms well, as expected.

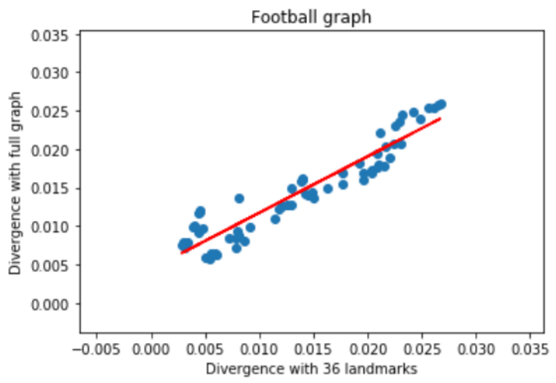

Figure 9: College Football Graph exhibits high correlation between the original divergence score and its approximated counterpart.

We compared the two sets of divergence scores for all embeddings, the first set based on the original algorithm and the second one based on the approximated version. The two sets of scores (as well as their rankings) are highly correlated as indicated by the following two measures of similarity: Pearson's correlation of 0.941 for the divergence scores and Kendall-tau of 0.802 for the rankings. Having said that, the rankings that we obtained are not identical. In Figure 10, we show the best and worst scoring embeddings for the approximated divergence score based on landmarks which should be compared with Figure 5 that shows the same for the original divergence score. However, the conclusion is the same as before: embeddings that score high are of good quality wheres the ones that score low are of poor quality.

Our next experiment is with Email-Eu-core Network on 986 vertices. As before, we tested all available embeddings. Clearly, our approximation algorithm provides a trade-off between the speed and the accuracy of the obtained approximation—the more landmarks we use, the better approximation we get but the algorithm gets slower. The goal of this experiment is to investigate how sensitive the approximation is as a function of the number landmarks. Since the ranking was preserved in this example, we only compared the Pearson's correlation between the two sets of divergence scores of all embeddings, the first one computed for the original graph on $n = 986$ vertices, and the second one computed for the approximated variant on n' landmarks with $n' \geq 25$—see Figure 11. As expected, there is a high correlation between the two sets with a satisfying outcomes already around $\sqrt{n}$ landmarks.

In order to see how the approximated algorithm behaves on large graphs, our last experiments are concerned with relatively large instances of ABCD graphs, on $n = 10{,}000$ vertices and on $n = 100{,}000$ vertices. Whereas the smaller graph can be easily tested by the original algorithm, dealing with the larger graph seems impractical (we only tested it for $n' \leq 10{,}000 < n = 100{,}000$

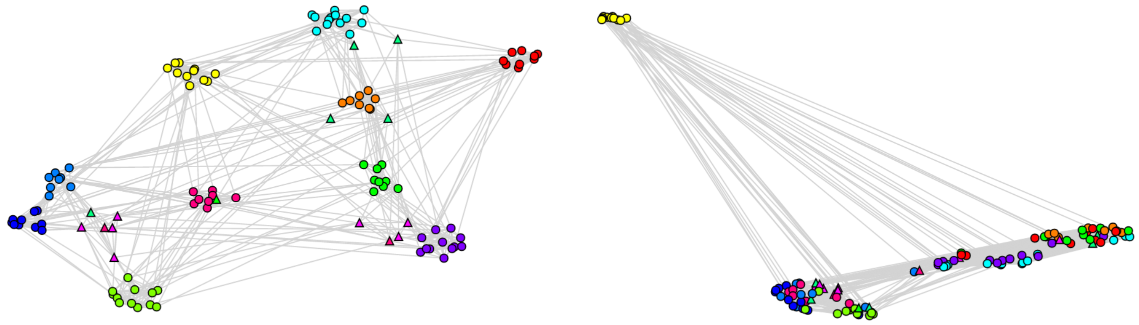

Figure 10: The College Football Graph. We show the best (left) and the worst (right) scoring embedding based on the approximated algorithm with landmarks.

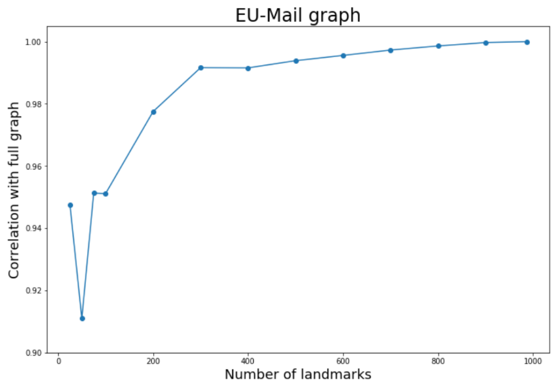

Figure 11: Pearson's correlation between the divergence scores computed for the original graph on $n = 986$ vertices and the ones computed for the approximated variant on $n' \geq 25$ landmarks.

landmarks). On the other hand, the approximated algorithm easily deals with graphs of that size.

For each graph, we tested the embedding obtained by 8-dim node2vec algorithm (time required to generate embedding for the small graph was roughly 32 seconds wheres the large graph required 6 minutes and 20 seconds to be processed). The results are presented in Figure 12. For each number of landmarks n', we plot the approximated divergence score as well as the time required to compute it on 2.2GHz Intel Xeon E5 processor. We clearly see the trade-off between the accuracy and the speed of the algorithm with the "sweet spot" around $n' \approx \sqrt{n}$ where the approximated divergence score is very close to the original divergence score whereas the algorithm is still extremely fast. In order to reach that conclusion, we tested the algorithm for the values of n' up to $n' = 10^4 = (10^5)^{4/5} = n^{4/5}$, much larger than $n' = \sqrt{n}$. As a result, in practice, one can easily deal with graphs or order $n = (10^4)^2 = 10^8$.

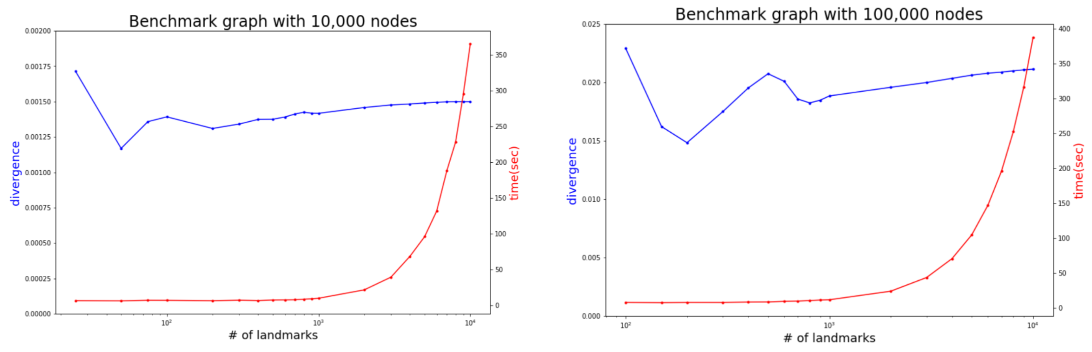

Figure 12: Comparing quality and speed for ABCD graphs. We compare the approximated divergence score and the time required to compute it as a function of the number of landmarks.

6 Future Directions

In this paper, our aim was to introduce a general framework for evaluating embeddings. This exploratory research showed that our divergence score is a very promising distinguisher. The next natural step is to do extensive experiments of various embedding algorithms on large real-world datasets in order to evaluate and compare them.

A further extension of this work could be made to weighted graphs or hypergraphs that are natural generalizations of graphs in which a single (hyper)edge can connect any number of vertices. As a consequence, hypergraphs are often more suitable and useful for representing and modelling many important networks and processes. Typical applications are related to social data analysis and include situations such as exchanging emails with several recipients, co-authorship of research papers, or analyzing security vulnerabilities of information networks. We are interested in generalizing classic notions to hypergraphs, such as clustering via modularity [43], as well as developing new algorithms to apply to them [44, 45]. Hence, a natural line of development of the proposed embedding comparison framework is to generalize it to allow for evaluation of embeddings of hypergraphs.

As a side effect of our research on evaluating graph embeddings, we have introduced the Geometric Chung-Lu model that is interesting on its own right and potentially applicable in other problems. As it is not the main focus of this paper, we did not analyze its graph-theoretic properties in detail. The properties, that are standard in the analysis of other models of complex networks, include clustering coefficient, connectivity, average path length, centrality measures. Their study remains as a subject for further research.

References

[1] B. Kamiński, P. Prałat, and F. Théberge, An Unsupervised Framework for Comparing Graph Embeddings, Journal of Complex Networks, in press, 27pp.

[2] Newman M. Networks: An Introduction. Oxford University Press; 2010.

[3] Bianconi G. Interdisciplinary and physics challenges of network theory. EPL. 2015; 111(5):56001.

[4] Hoff PD, Raftery AE, Handcock MS. Latent space approaches to social network analysis, J. Amer. Stat. Assoc. 2002; 97(460) 1090-1098.

[5] Krioukov D. Clustering means geometry in networks. Phys Rev Lett. 2016; 208302(May):1-5.

[6] Zuev K, Boguna M, Bianconi G, Krioukov D. Emergence of Soft Communities from Geometric Preferential Attachment. Scientific Reports. 2015; 5,9421.

[7] Gastner MT, Newman MEJ. The spatial structure of networks. European Physical Journal B. 2006; 49(2):247-252.

[8] Expert P, Evans TS, Blondel VD, Lambiotte R. Uncovering space-independent communities in spatial networks. Proceedings of the National Academy of Sciences. (2011); 108(19):7663-7668.

[9] J. Janssen. Spatial Models for Virtual Networks. CiE 2010, LNCS 6158, pp. 201-210, 2010.

[10] Higham DJ, Rasajski M, Przulj N. Fitting a geometric graph to a protein-protein interaction network. Bioinformatics. 2008; 24(8):1093-1099.

[11] Poulin V., Théberge F. (2019) Ensemble Clustering for Graphs. In: Aiello L., Cherifi C., Cherifi H., Lambiotte R., Lió P., Rocha L. (eds) Complex Networks and Their Applications VII. COMPLEX NETWORKS 2018. Studies in Computational Intelligence, vol 812. Springer, Cham.

[12] Lin J. Divergence measures based on the shannon entropy. In: IEEE Transactions on Information theory, 37(1), pp.145-151 (1991)

[13] Waxman, B. M. (1988). Routing of multipoint connections. IEEE journal on selected areas in communications, 6(9), 1617-1622.

[14] Kosmidis, K., Havlin, S., Bunde, A. (2008). Structural properties of spatially embedded networks. EPL (Europhysics Letters), 82(4), 48005.

[15] L. Barnett, E. Di Paolo, S. Bullock, Spatially embedded random networks, Phys. Rev. E, vol. 76, Nov. 2007.

[16] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá. Hyperbolic geometry of complex networks. Phyical Review E, 82:036106, 2010.

[17] W. Aiello, A. Bonato, C. Cooper, J. Janssen, and P. Pralat, A spatial web graph model with local influence regions, Internet Mathematics 5 (2009), 175-196.

[18] M. Deijfen, R. van der Hofstad, and G. Hooghiemstra. Scale-free percolation. Annales de l'Institut Henri Poincare, Probabilites et Statistiques, 49(3):817–838, 2013.

[19] Bringmann, K., Keusch, R., and Lengler, J. (2015). Sampling geometric inhomogeneous random graphs in linear time. arXiv preprint arXiv:1511.00576.

[20] Chung FRK, Lu L. Complex Graphs and Networks. American Mathematical Society; 2006.

[21] Seshadhri C, Kolda TG, Pinar A. Community structure and scale-free collections of Erdös-Rényi graphs. Physical Review E. 2012; 85: 056109.

[22] Kolda TG, Pinar A, Plantenga T, Seshadhri C. A scalable generative graph model with community structure. SIAM Journal on Scientific Computing. 2014; 36: C424–C452.

[23] Winlaw M, DeSterck H, Sanders G. An In-Depth Analysis of the Chung-Lu Model. Lawrence Livermore Technical Report LLNL-TR-678729. 2015; doi: 10.2172/1239211.

[24] Norros, I., Reittu, H. (2006). On a conditionally Poissonian graph process. Advances in Applied Probability, 38(1), 59-75.

[25] Z. Lu, J. Wahlström, A. Nehorai, Community Detection in Complex Networks via Clique Conductance. Nature Scientific Reports (2018) 8:5982.

[26] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. Phys. Rev. E, 78(4), 2008.

[27] B. Kamiński, P. Prałat, and F. Théberge, Artificial Benchmark for Community Detection (ABCD) — Fast Random Graph Model with Community Structure, pre-print arXiv:2002.00843, 2020.

[28] Zachary, W. W. An information flow model for conflict and fission in small groups. Journal of Anthropological Research 33, 452-473 (1977).

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

[30] J. Leskovec, A. Krevl. SNAP Datasets: Stanford Large Network Dataset Collection, http://snap.stanford.edu/data.

[31] A.L. Barabasi. Network Science. Cambridge U Press, 2016.

[32] A. Grover, J. Leskovec. node2vec: Scalable Feature Learning for Networks. KDD 2016: 855–864.

[33] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In Proceedings of the 2018 World Wide Web Conference (WWW'18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 539-548.

[34] Avrachenkov, K., Chebotarev, P., Rubanov, D. (2017). Kernels on Graphs as Proximity Measures. In International Workshop on Algorithms and Models for the Web-Graph (pp. 27-41). Springer. (conference version)

[35] Avrachenkov, K., Chebotarev, P., Rubanov, D. (2019). Similarities on graphs: Kernels versus proximity measures. European Journal of Combinatorics, 80, 47-56. (journal version)

[36] Schoenberg, I. J. (1938). Metric spaces and completely monotone functions. Annals of Mathematics, 811-841.

[37] Schoenberg, I. J. (1938). Metric spaces and positive definite functions. Transactions of the American Mathematical Society, 44(3), 522-536.

[38] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings 24th International Conference on World Wide Web, 2015, pp. 1067–1077.

[39] L. McInnes, J. Healy, J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. pre-print arXiv:1802.03426, 2018.

[40] Poulin V., Théberge F. Ensemble Clustering for Graphs: Comparison and Applications., pre-print, arXiv:1903:08012, 2019.

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

[42] Rosvall M, and Bergstrom C.T., Maps of random walks on complex networks reveal community structure, PNAS 105 (1118), 2008.

[43] B. Kaḿinski, V. Poulin, P. Prałat, P. Szufel, and F. Théberge, Clustering via Hypergraph Modularity, PLoS ONE 14(11): e0224307.

[44] A. Antelmi, G. Cordasco, B. Kamiński, P. Prałat, V. Scarano, C. Spagnuolo, and P. Szufel, SimpleHypergraphs.jl — Novel Software Framework for Modelling and Analysis of Hypergraphs, Proceedings of the 16th Workshop on Algorithms and Models for the Web Graph (WAW 2019), Lecture Notes in Computer Science 11631, Springer, 2019.

[45] A. Antelmi, G. Cordasco, B. Kamiński, P. Prałat, V. Scarano, C. Spagnuolo, and P. Szufel, Analyzing, Exploring, and Visualizing Complex Networks via Hypergraphs using SimpleHypergraphs.jl, Internet Mathematics (2020), 32pp.

A Appendix — Geometric Chung-Lu Model is Well-defined

Let us state the problem in a slightly more general situation. Suppose that for each pair $i, j \in [n]$ we have $a_{ij} = a_{ji} \in \mathbb{R}_+$, and for each $i \in [n]$ we have $a_{ii} = 0$ and $b_i \in \mathbb{R}_+$. In our application, elements of vector $\mathbf{b} = (b_i)_{i\in[n]}$ satisfy $b_i = w_i \in [n-1]$ and correspond to the degree distribution, and elements of matrix A satisfy $a_{i,j} = g(d_{i,j}) \in (0, 1]$ and correspond to the embedding distance between vertices. Our goal is to investigate if there is a solution, $x_i \in \mathbb{R}_+$ for $i \in [n]$, of the following system of equations:

$$b_i = x_i \sum_{j=1}^{n} a_{ij} x_j \qquad \text{for all } i \in [n]. \tag{7}$$

If there is one, then is this solution unique?

The $n = 2$ case a degenerate case that exhibit a different behaviour but is easy to investigate: $b_1 = x_1 x_2 a_{12}$ and $b_2 = x_2 x_1 a_{21}$. It follows that the solution exists if and only if $b_1 = b_2$. If this is the case, then there are infinite number of solutions, each of them of the form $(x_1, x_2) = (t, b_1/(a_{12}t))$ for some $t \in \mathbb{R}_+$.

Suppose now that $n \geq 3$. We will show that the desired solution of (7) exists if and only if

$$\sum_{i=1}^{n} b_i > 2 \max_{i\in[n]} b_i. \tag{8}$$

In other words, the condition is that the maximum element in vector $\mathbf{b}$ is smaller than the sum of the other elements. This is a very mild condition that holds in our application. More importantly, this solution is unique.

We will start with proving uniqueness. After that, we will show that (8) is indeed an obvious, necessary condition before proving that it is also a sufficient one.

A.1 Uniqueness

Let us assume that $n \geq 3$. For a contradiction, suppose that we have two different solutions: $\mathbf{x} = (x_i)_{i\in[n]}$ ($x_i \in \mathbb{R}_+$, $i \in [n]$) and $\mathbf{y} = (y_i)_{i\in[n]}$ ($y_i \in \mathbb{R}_+$, $i \in [n]$). It follows that for all $i \in [n]$ we have

$$b_i = f_i(\mathbf{x}) = f_i(\mathbf{y}), \qquad \text{where } f_i(\mathbf{x}) = x_i \sum_{j=1}^{n} a_{ij} x_j.$$

Let us analyze what happens at point $\mathbf{z} = t\mathbf{x} + (1-t)\mathbf{y}$ for some $t \in [0,1]$ (that is, $z_i = tx_i + (1-t)y_i$, $i \in [n]$). For each $i \in [n]$ we get

$$\begin{aligned}
f_i(\mathbf{z}) &= (tx_i + (1-t)y_i) \sum_{j=1}^{n} a_{ij}(tx_j + (1-t)y_j) \\
&= \sum_{j=1}^{n} a_{ij} \left(t^2 x_i x_j + t(1-t)(x_i y_j + x_j y_i) + (1-t)^2 y_i y_j \right) \\
&= f(\mathbf{x})t^2 + \frac{t(1-t)}{x_i y_i}(f(\mathbf{y})x_i^2 + f(\mathbf{y})y_i^2) + f(\mathbf{y})(1-t)^2 \\
&= b_i \left(t^2 + \frac{t(1-t)}{x_i y_i}(x_i^2 + y_i^2) + (1-t)^2 \right) \\
&= b_i \left(1 - 2t(1-t) + \frac{t(1-t)}{x_i y_i}(x_i^2 + y_i^2) \right) \\
&= b_i \left(1 + \frac{t(1-t)}{x_i y_i}(x_i^2 - 2x_i y_i + y_i^2) \right) \\
&= b_i \left(1 + t(1-t)\frac{(x_i - y_i)^2}{x_i y_i} \right) =: h_i(t).
\end{aligned}$$

Note that $h_i'(1/2) = 0$ for all i (as either $x_i - y_i$ vanishes and so $h_i(t)$ is a constant function or it does not vanish but then $h_i(t)$ is a parabola with a maximum at $t = 1/2$). For convenience, let $\mathbf{v} = (\mathbf{x}+\mathbf{y})/2$ and $\mathbf{s} = (\mathbf{x}-\mathbf{y})/2$ (that is, $v_i = (x_i + y_i)/2$ and $s_i = (x_i - y_i)/2$ for all $i \in [n]$). It follows that

$$\frac{df_i}{dh}\Big(\mathbf{v} + h\mathbf{s} \mid h = 0 \Big) = 0.$$

On the other hand,

$$f_i(\mathbf{v} + h\mathbf{s}) = \sum_{j=1}^{n} a_{ij}(v_i + hs_i)(v_j + hs_j)$$

so

$$\frac{df_i}{dh}(\mathbf{v} + h\mathbf{s}) = \sum_{j=1}^{n} a_{ij}(s_j(v_i + hs_i) + s_i(v_j + hs_j)).$$

Combining the two observations, we get that

$$0 = \frac{df_i}{dh}\Big(\mathbf{v} + h\mathbf{s} \mid h = 0 \Big) = \sum_{j=1}^{n} a_{ij}(s_j v_i + s_i v_j) = s_i \sum_{j=1}^{n} a_{ij} v_j + \sum_{j=1}^{n} s_j a_{ij} v_i.$$

Since $a_{ii} = 0$, it is equivalent to

$$\mathbf{g}_i \; \mathbf{s} = 0,$$

where $\mathbf{g}_i$ is a row vector such that $g_{ij} = a_{ij} v_i$ if $i \neq j$ and $g_{ii} = \sum_{j=1}^{n} a_{ij} v_j$. Since this is true for all $i \in [n]$, we get

$$G\mathbf{s} = \mathbf{0},$$

where g_{ij}'s, the elements of matrix G, are defined as above. Since our assumption is that $\mathbf{x} \neq \mathbf{y}$, we have that $\|s\| > 0$ and so we get that $\det(G) = 0$.

Now, as multiplying a column of a matrix by a positive constant does not change the sign of its determinant, we may multiply column j by $v_j > 0$ to conclude that

$$\det(C) = 0,$$

where $c_{ii} = \sum_{j=1}^{n} a_{ij} v_i v_j$ and $c_{ij} = a_{ij} v_i v_j$ for $i \neq j$. Thus, C is a symmetric matrix whose diagonal is a sum of all its entries in a row off diagonal.

For a further reference note that $C = GZ$, where Z is a diagonal matrix where $\forall j \in [n]$: $z_{jj} = v_j$ and $\forall i, j \in [n], i \neq j : z_{ij} = 0$. As $\forall j \in [n] : v_j > 0$ we get that $\det(Z) = \prod_{j=1}^{n} v_j > 0$. Therefore, as $\det(C) = \det(GZ) = \det(G)\det(Z)$, we have that $\det(C)$ and $\det(G)$ have the same sign.

Now, since $\det(C) = 0$, we get that there exists a non-zero vector $\mathbf{u}$ such that $C\mathbf{u} = \mathbf{0}$. Without loss of generality, we may assume that the first entry of this vector is a largest one in absolute value. Since we may scale this vector, without loss of generality, we may also assume that $\|\mathbf{u}\|_\infty = 1$; that is, $u_1 = 1$ and $|u_i| \leq 1$ for all i. Let us consider the product of the 1st row of C and vector $\mathbf{u}$. We have that

$$0 = 1 \cdot \sum_{j=2}^{n} a_{1j} v_1 v_j + \sum_{j=2}^{n} u_j a_{1j} v_1 v_j = \sum_{j=2}^{n} (1 + u_j) a_{1j} v_1 v_j.$$

But this means that $u_j = -1$ for $j > 1$ (since all other numbers involved are positive). Finally, by considering the 2nd row of the product $C\mathbf{u} = \mathbf{0}$, we get that

$$0 = a_{21} v_2 v_1 - \sum_{j \neq 2} a_{2j} v_2 v_j - \sum_{j=3}^{n} a_{2j} v_2 v_j = -2 \sum_{j=3}^{n} a_{2j} v_2 v_j.$$

This is possible only if $n = 2$; that is, when the sum is trivial. Since $n \geq 3$, we get the desired contradiction. This finishes the proof of the uniqueness of the solution of (7) for $n \geq 3$.

Also note that G is the Jacobian matrix of the function $[f_i(\mathbf{x})]$ and from the proof it follows that $\det(G) \neq 0$ in all admissible points where the system of equations $b_i = f_i(\mathbf{x})$ has a solution, as $\det(G) = \det(C)/\det(Z)$, and we have just proven that $\det(C) \neq 0$.

A.2 Necessity

Without loss of generality, we may assume that the sequence of b_i's is non-increasing. In particular, b_1 is a largest value. Suppose that $x_i \in \mathbb{R}_+$, $i \in [n]$, is a solution of the system (7). Since $a_{ij} = a_{ji} > 0$ for $i \neq j$ and $a_{ii} = 0$, we observe that for $n \geq 3$

$$\sum_{i=2}^{n} b_i = \sum_{i=2}^{n} \left(x_i \sum_{j=1}^{n} a_{ij} x_j \right) > \sum_{i=2}^{n} (x_i a_{i1} x_1) = x_1 \sum_{i=1}^{n} a_{1i} x_i = b_1.$$

Hence, we get that (8) is a necessary condition for the existence of the solution.

A.3 Sufficiency

In this subsection, we will still assume that $n \geq 3$. For a contradiction, suppose that there exists a vector $\mathbf{b} = (b_i)_{i \in [n]}$, with $b_i > 0$ for all i, that satisfies (8) but for which there is no solution to the system (7). We will call such vectors *infeasible*; on the other hand, vectors that yield a solution $\mathbf{x} = (x_i)_{i\in[n]}$, with $x_i > 0$ for all i, will be called *feasible* (as proved above, the solution must be unique). Without loss of generality, we may assume that b_1 is a largest value in vector $\mathbf{b}$.

Let us now construct another vector $\mathbf{b}' = (b'_i)_{i\in[n]}$ for which there is a solution to (7) (that is, $\mathbf{b}'$ is feasible) but also $b'_1 = b_1$ is a largest element in $\mathbf{b}'$. Indeed, it can be done easily by, for example, taking $x'_1 = s$ for some large enough $s \in \mathbb{R}_+$ and $x'_2 = \ldots = x'_n = b_1/(s\sum_{j\in[n]} a_{1j})$. We immediately get that

$$b'_1 = x'_1 \sum_{j\in[n]} a_{1j}x'_j = b_1$$

and for $i \geq 2$ we have

$$b'_i = x'_i \sum_{j\in[n]} a_{ij}x'_j = x'_i a_{i1} x'_1 + x'_i \sum_{j=2}^{n} a_{ij}x'_j = b_1 \frac{a_{1i}}{\sum_{j\in[n]} a_{1j}} + \left(\frac{b_1}{s\sum_{j\in[n]} a_{1j}}\right)^2 \sum_{j=2}^{n} a_{ij} < b_1,$$

provided that $s \in \mathbb{R}_+$ is sufficiently large (since $n \geq 3$, we get that $a_{1i}/\sum_{j\in[n]} a_{1j} < 1$, and the second term above tends to zero as $s \to \infty$).

We will consider points along the line segment between $\mathbf{b}'$ and $\mathbf{b}$, namely,

$$\mathbf{b}(t) = (b_i(t))_{i\in[n]} = (1-t)\mathbf{b}' + t\mathbf{b}, \qquad \text{for } t \in [0,1].$$

Since $\mathbf{b}'$ is feasible and we already proved that (8) is a necessary condition, $\mathbf{b}'$ satisfies (8). But, not only $\mathbf{b}$ and $\mathbf{b}'$ satisfy it but also $\mathbf{b}(t)$ satisfies it for any $t \in [0,1]$. Let us fix any $t \in (0,1)$. Clearly, $b_1(t) = b_1 = b'_1$ is a largest value in $\mathbf{b}(t)$. More importantly,

$$\sum_{i\in[n]} b_i(t) = (1-t)\sum_{i\in[n]} b'_i + t\sum_{i\in[n]} b_i > (1-t)\cdot 2\max_{i\in[n]} b'_i + t\cdot 2\max_{i\in[n]} b_i = 2b_1(t) = 2\max_{i\in[n]} b_i(t)$$

and so, indeed, (8) holds for $\mathbf{b}(t)$. It follows that there exists a universal $\varepsilon > 0$ such that for any $t \in [0,1]$ we have

$$2\max_{i\in[n]} b_i(t) < (1-\varepsilon)\sum_{i\in[n]} b_i(i). \tag{9}$$

Fix $t \in (0,1)$ and suppose that $\mathbf{b}(t)$ is feasible. Let $\mathbf{x}(t) = (x_i(t))_{i\in[n]}$ be the (unique) solution for $\mathbf{b}(t)$. Let $G(t) = \left(\frac{\partial f_i}{\partial x_j}\right)_{ij}$ be the Jacobian matrix of our transformation function at point $\mathbf{x}(t)$; that is, $g_{ij}(t) = a_{ij}x_i(t)$ if $i \neq j$ and $g_{ii}(t) = \sum_{j\in[n]} a_{ij}x_j(t)$. From the analysis performed in the proof of uniqueness of the solution it follows that $\det(G(t)) \neq 0$ and so our transformation is a local diffeomorphism. As a result, any open set in $\mathbb{R}^n$ containing $\mathbf{x}(t)$ is mapped to an open set in $\mathbb{R}^n$ containing $\mathbf{b}(t)$. In particular, there exists $\delta > 0$ such that $\mathbf{b}(s)$ is feasible for any $t - \delta \leq s \leq t + \delta$. Combining this observation with the fact that $\mathbf{b}$ is *not* feasible we get that there exists $T \in (0,1]$ such that $\mathbf{b}(T)$ is not feasible but $\mathbf{b}(t)$ is feasible for any $t \in [0,T)$.

Consider then any sequence $(t_i)_{i\in\mathbb{N}}$ of real numbers $t_i \in [0,T)$ such that $t_i \to T$ as $i \to \infty$; for example, $t_i = T(1-1/i)$. All limits from now on will be for $i \to \infty$. Recall that $\mathbf{b}(t_i)$ is feasible and so $\mathbf{x}(t_i)$ is well-defined.

Let us first note that it is impossible that $\|\mathbf{x}(t_i)\|$ is bounded for infinitely many i. Indeed, if it were the case, then by Bolzano-Weierstrass theorem it would have a subsequence $(\mathbf{x}(t_{s_i}))_{i\in[n]}$ such that $\|\mathbf{x}(t_{s_i})\| \to c \in \mathbb{R}$. However, then, by continuity of our transformation, the limit value $\mathbf{b}(T)$ would be feasible. Note, in particular, that it is impossible that $x_j(t_{s_i}) \to 0$ for some j as then the corresponding $b_j(t_{s_i})$ would also tend to zero which implies that $b_j(T) = 0$; this is not possible as both b_j and b'_j are bounded away from zero.

It follows that $\|\mathbf{x}(t_i)\| \to \infty$. This means that there exist $p \in [n]$ and subsequence $(t_{s_i})_{i\in\mathbb{N}}$ such that $x_p(t_{s_i}) \to \infty$. Let us now observe that it is not possible that $x_p(t_{s_i}) \to \infty$ and $x_q(t_{s_i})$ does not tend to zero for some $q \neq p$. Indeed, since $b_p(t_{s_i}) \geq a_{pq}x_q(t_{s_i})x_p(t_{s_i})$, this would imply that for any constant c, $b_p(t_{s_i}) > c$ for an infinite number of i's. This is impossible as $b_p(t_{s_i}) \to b_p(T) \leq b_1(T)$. We get that $x_p(t_{s_i}) \to \infty$ for precisely one index p and $x_q(t_{s_i}) \to 0$ for all $q \neq p$.

Now, for a given $q \neq p$, note that

$$\sum_{j\neq p} a_{qj}x_q(t_{s_i})x_j(t_{s_i}) \to 0$$

and

$$a_{qp}x_q(t_{s_i})x_p(t_{s_i}) + \sum_{j\neq p} a_{qj}x_q(t_{s_i})x_j(t_{s_i}) = \sum_{j\in[n]} a_{qj}x_q(t_{s_i})x_j(t_{s_i}) = b_q(t_{s_i}) \to b_q(T).$$

We get that for sufficiently large values of i, $a_{qp}x_q(t_{s_i})x_p(t_{s_i}) > (1-\varepsilon/2)b_q(T)$, where $\varepsilon > 0$ is the same as in (9). It follows that for all sufficiently large i,

$$b_p(t_{s_i}) = \sum_{q\neq p} a_{pq}x_p(t_{s_i})x_q(t_{s_i}) \geq \left(1 - \frac{\varepsilon}{2}\right)\sum_{q\neq p} b_q(T)$$

and so also in the limit

$$b_p(T) \geq \left(1 - \frac{\varepsilon}{2}\right)\sum_{q\neq p} b_q(T).$$

But this means that

$$2\max_{i\in[n]} b_i(T) \geq 2b_p(T) \geq b_p(T) + \left(1 - \frac{\varepsilon}{2}\right)\sum_{q\neq p} b_q(T) \geq \left(1 - \frac{\varepsilon}{2}\right)\sum_{q\in[n]} b_q(T)$$

that contradicts (9). Hence, we get that (8) is also a sufficient condition for the existence of the solution.

B Appendix — Generalized Geometric Chung-Lu Model is Well-defined

Our goal is to analyze the generalized model for which $a_{ii} > 0$ for some $i \in [n]$. Without loss of generality, we may assume that $b_1 \geq b_i$ for $i \in [n]$. As in the original model, the case $n = 2$ is a degenerate case which behaves differently and so we treat it independently.

For the $n = 2$ case, we have the following system of equations:

$$\begin{aligned} b_1 &= a_{11}x_1^2 + a_{12}x_1x_2 \\ b_2 &= a_{22}x_2^2 + a_{12}x_1x_2. \end{aligned}$$

We will independently consider the following 3 sub-cases.

- $a_{11} = 0$ (and so $a_{22} > 0$): Since it is assumed that $x_2 > 0$, $b_2 = a_{22}x_2^2 + b_1 > b_1$. But this contradicts the fact that $b_1 \geq b_2$. It follows that the system does not have any solution.

- $a_{11} > 0$ and $a_{22} > 0$: Let $f(x_1, x_2) := a_{22}x_2^2 + a_{12}x_1x_2$. Let us first note that for any given $x_1 > 0$, x_2 is uniquely determined by the first equation, namely, $x_2 = (b_1 - a_{11}x_1^2)/(a_{12}x_1)$. In particular, if $x_1 = \sqrt{b_1/a_{11}}$, then $x_2 = 0$ and so $f(x_1, x_2) = 0$. Moreover, as we decrease x_1, we continuously increase both x_2 and $f(x_1, x_2) = a_{22}x_2^2 + b_1 - a_{11}x_1^2$. Finally, if $x_1 \to 0$, then $f(x_1, x_2) \to \infty$. After combining all of these observations, we get that there exists a unique solution (x_1, x_2) to $f(x_1, x_2) = b_2$.

- $a_{22} = 0$ (and so $a_{11} > 0$): We have that $x_1 = \sqrt{(b_1 - b_2)/a_{11}}$, which is a unique non-negative solution, provided that $b_1 > b_2$.

We can summarize the result along with the original theorem for $n = 2$ as follows: the system has a solution if and only if

$$(a_{11} > 0 \wedge (a_{22} > 0 \vee b_1 > b_2)) \vee (a_{11} = a_{22} = 0 \wedge b_1 = b_2),$$

and the solution is unique except for the case when $a_{11} = a_{22} = 0$.

For the $n \geq 2$ case, we consider 2 sub-cases depending on whether $a_{11} = 0$ or not. If $a_{11} = 0$, then the properties are exactly the same as in the original model. Indeed, it is easy to see that both necessary and sufficient conditions are not affected. The proof of uniqueness is the same as before except that this time $g_{ii} = a_{ii}v_i + \sum_{j=1}^{n} a_{ij}v_j$. The other ingredients of the proof are the same as before: after considering row 1 and then row 2 of the product $C\mathbf{u} = \mathbf{0}$ we conclude that $\det C \neq 0$, and if solution exists then it is unique.

Let us now assume that $a_{11} > 0$ and start with the uniqueness of the solution, if it exists. If $a_{ii} = 0$ for some i, then we proceed as in the original proof but first consider row i and then row 1. On the other hand, if $a_{ii} > 0$ for all i, then $\det C > 0$ as all eigenvalues of C are positive following Gershgorin circle theorem (since C is symmetric).

We will now prove that the considered system has a solution for any vector $\mathbf{b}$ if $a_{11} > 0$. For a contradiction, suppose that there exists some $\mathbf{b}$ for which the system does not have a solution. Then, as in the original proof, we construct a feasible vector $\mathbf{b}'$ by considering s very close but below $\sqrt{b_1/a_{11}}$ and take $x_2' = \ldots = x_n' = (b_1 - a_{11}s^2)/(s\sum_{j\in[n]} a_{1j})$. Arguing as before, we get that $b_i' < b_1$ for $i > 1$. The rest of the proof remains the same and we conclude that $x_p(t_{s_i}) \to \infty$. This implies that $a_{pp} = 0$, as otherwise x_p would be bounded since $b_p > a_{pp}x_p^2$. Finally, from the fact that $a_{pp} = 0$ we get the desired contradiction in exactly the same way as in the original proof.

In summary, for $n \geq 3$ we get the following condition for the existence and the uniqueness of the solution of the system:

$$a_{11} > 0 \vee 2b_1 < \sum_{i=1}^{n} b_i.$$

C More Illustrations of the GCL Model

In Section 2, we illustrated our framework with the small Zachary's Karate Club graph. In this section, we provide the same illustration for all the other graphs we considered except for the larger email graph which is less amenable to nice visualization.

For each graph considered, we select the best scoring embedding we found, and we plot the divergence score as a function of the α-parameter that we use in our framework. Recall that we search over a grid of values for α and we select the one for which the divergence score in (3) is minimized. We also plot the graph itself using this best embedding.

The Geometric Chung-Lu model computes probabilities of seeing edges between vertices as a function of the degrees of the corresponding vertices and their distance in embedded space, the latter being a function of α. Thus, we can use this model to generate random graphs given the degrees of vertices and some embedding. With small values of α, the importance of the distances is small (at the extreme case, when $\alpha = 0$, it is completely neglected), and several long edges (between communities) are generated. With large values of α, it is the opposite and we expect much fewer edges between communities. The optimal value, which we denote $\hat{\alpha}$, aims at preserving the right balance between long and short edges (respectively between and within communities).

For each graph considered, we show three generated random graph respectively using $\alpha = 0, \alpha = \hat{\alpha}$, and some value of $\alpha > \hat{\alpha}$.

C.1 The College Football Graph

For this graph, most communities are very strong, and the best embedding puts their vertices quite close, as we see in Figure 13. The impact of the choice of α for the distance function in embedded space is clearly seen in the bottom row of Figure 13. With $\alpha = 0$, there are many edges between communities, as only the expected degrees are preserved. For α larger than the optimal value, we see the opposite behaviour. The random graph generated with the optimal α value is visually very similar to the real graph shown in the top row.

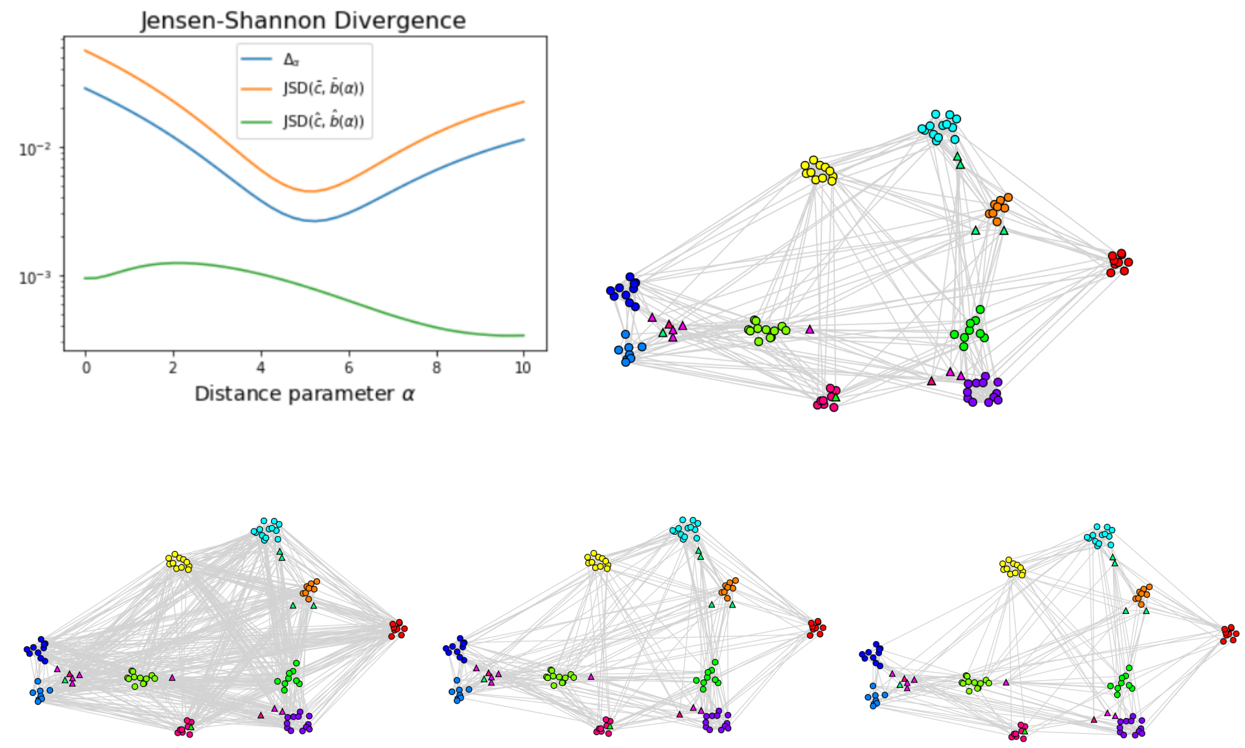

Figure 13: The College Football Graph. On the top row, we illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). In the bottom row, we generate edges using the Geometric Chung-Lu Model respectively with $\alpha = 0$ (left), the optimal value $\hat{\alpha} = 5.25$ (center) and $\alpha = 7$ (right).

C.2 LFR Graph with Strong Communities

The communities are very strong given the choice of value $\mu = 0.15$, so good embeddings should put their vertices quite close, as we see in Figure 14. The impact of the choice of α for the distance function in embedded space is clearly seen in the bottom row of Figure 14, with the same conclusion as for the College Football graph.

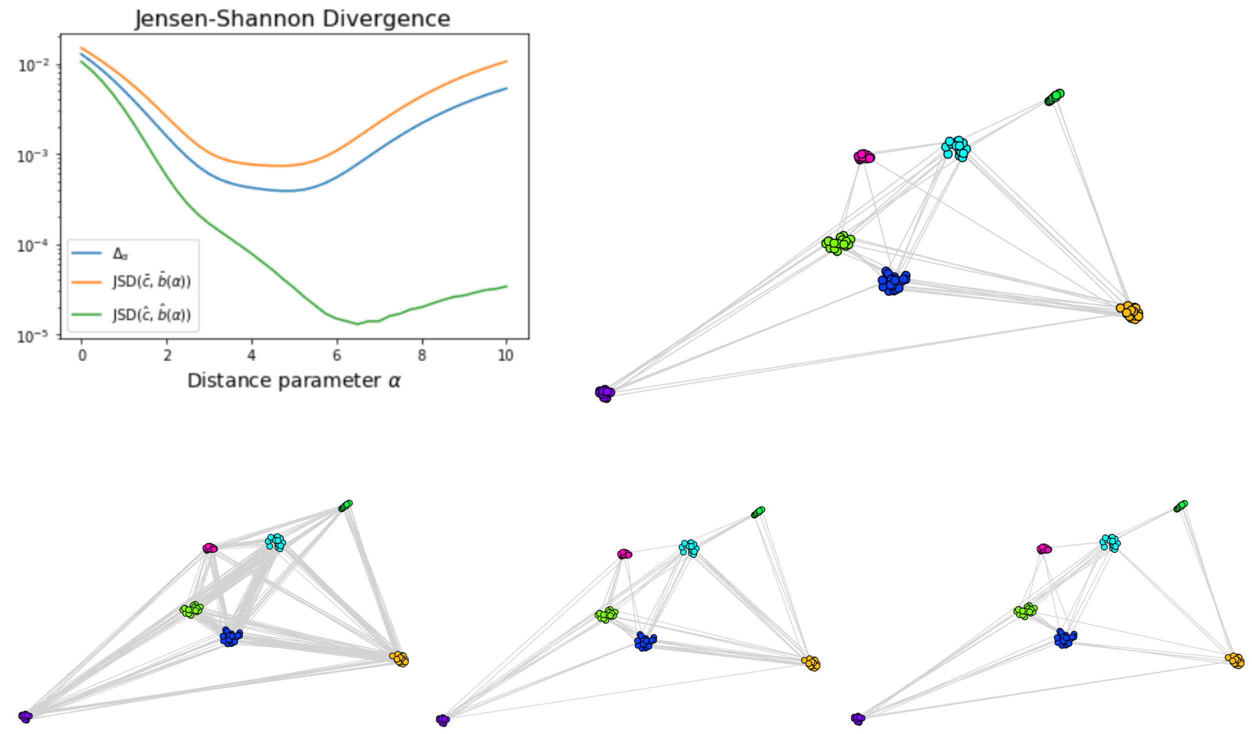

Figure 14: For the LFR graph with mixing parameter $\mu = 0.15$, on the top row, we illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). On the bottom row, we generate edges via the Geometric Chung-Lu Model with distance parameters $\alpha = 0$ (left), the optimal value $\hat{\alpha} = 4.75$ (center) and $\alpha = 7$ (right).

C.3 LFR Graph with Weaker Communities

Results for the LFR graph with mixing parameter $\mu = 0.35$ are shown in Figure 15. For the best embedding, vertices within communities are not as close as in the previous LFR graph, as expected. Conclusions regarding the values of α are the same as for the previous examples.

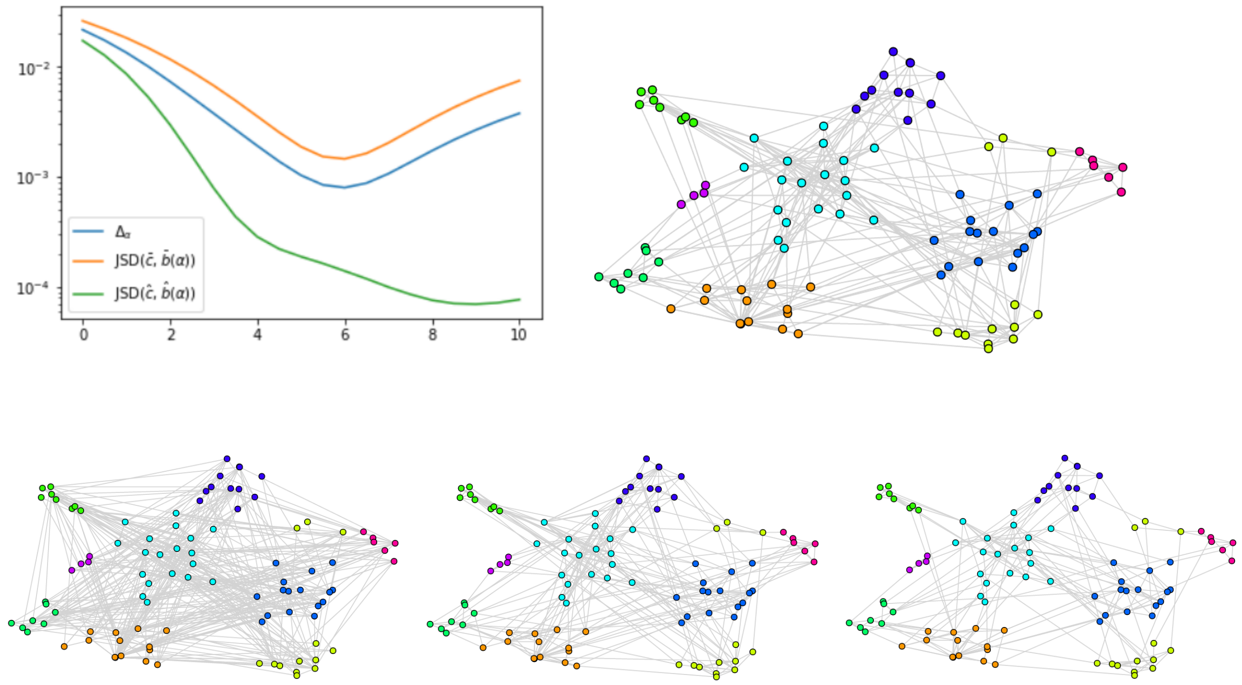

Figure 15: For the LFR graph with mixing parameter $\mu = .35$, on the top row, we illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). On the bottom row, we generate edges via the Geometric Chung-Lu Model with distance parameters $\alpha = 0$ (left), the optimal value $\hat{\alpha} = 6$ (center) and $\alpha = 10$ (right).

C.4 Noisy LFR Graph

In this graph, LFR graph with mixing parameter $\mu = 0.55$, for each community, there are more expected external edges than internal edges. Nevertheless, in the best embedding shown in Figure 16, we still see some spatial grouping of the communities.

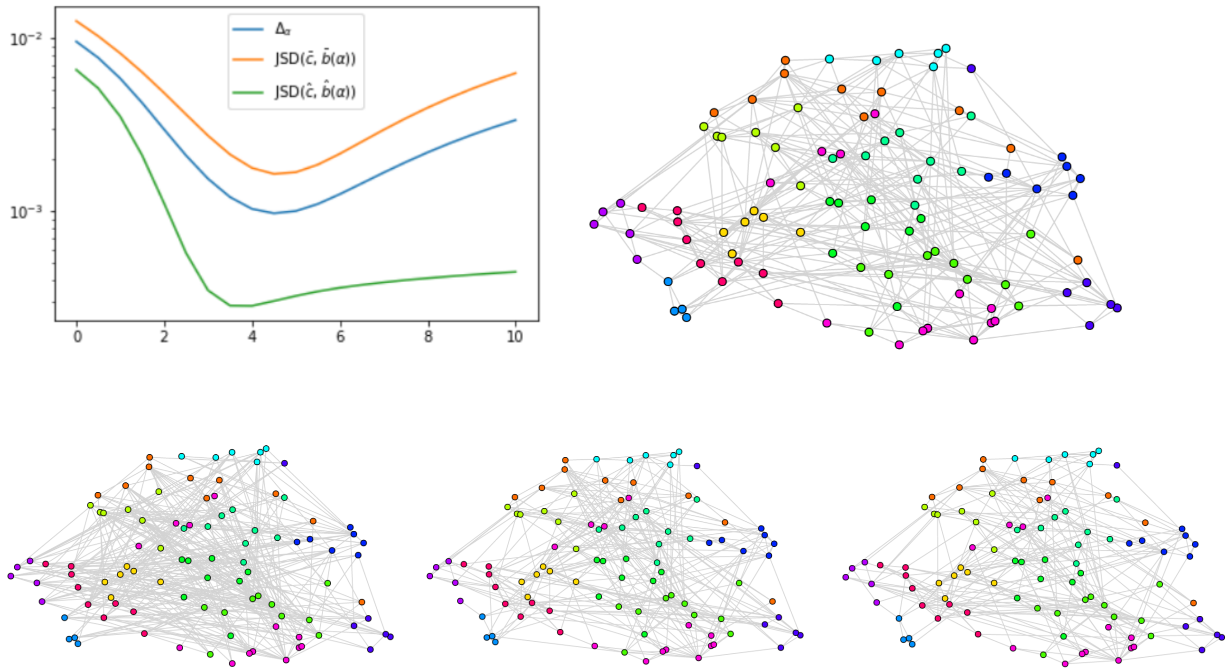

Figure 16: For the LFR graph with mixing parameter $\mu = 0.55$, on the top row, we illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). On the bottom row, we generate edges via the Geometric Chung-Lu Model with distance parameters $\alpha = 0$ (left), the optimal value $\hat{\alpha} = 4.5$ (center), and $\alpha = 7$ (right).