A Scalable Unsupervised Framework for Comparing Graph Embeddings

Bogumił Kamiński[1], Paweł Prałat[2], and François Théberge[3]

[1] Decision Analysis and Support Unit, SGH Warsaw School of Economics, Warsaw, Poland; bogumil.kaminski@sgh.waw.pl
[2] Department of Mathematics, Ryerson University, Toronto, ON, Canada; pralat@ryerson.ca
[3] Tutte Institute for Mathematics and Computing, Ottawa, ON, Canada; theberge@ieee.org

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. Unfortunately, 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. The complexity of the original algorithm 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. Moreover, we perform some detailed quality and speed benchmarks.

Keywords: Graph Embedding · Geometric Chung-Lu Model.

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, and connections between geometry and network clustering and community structure. 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. See [4] 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. 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, in [1] we experimented with synthetic networks as well as real-world networks, and various embedding algorithms. In this paper, we concentrate on the complexity challenges of the original algorithm and propose a fast approximated algorithm that works very well in practice.

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. In the recent paper [1], we experimented with many datasets and embedding algorithms to show that the framework works well. In this paper, for illustration purposes, we use some of these datasets and their corresponding embeddings. However, due to the space limitation, we do not explain how they are constructed. Interested reader is directed to [1] for more details. Here, we focus on improvements that were required to make an algorithm scalable. The results presented in Section 4 for a novel extension of the original algorithm are the main contribution of this paper. We conclude with a discussion on some future directions in Section 5.

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 was illustrated in [1], 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 $\arg\min_{i\in[m]} \Delta_{\mathcal{E}_i}(G)$ (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 [5], and is shown to have good stability.
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 $1 \le i < j \le \ell$, 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 $1 \le i < j \le \ell$, 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

$$\begin{aligned} \bar{\mathbf{b}}_{\mathcal{E}}(\alpha) &= (b_{1,2}, \ldots, b_{1,\ell}, b_{2,3}, \ldots, b_{2,\ell}, \ldots, b_{\ell-1,\ell}) \\ \hat{\mathbf{b}}_{\mathcal{E}}(\alpha) &= (b_1, \ldots, b_\ell) \end{aligned} \tag{2}$$

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 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 $\hat{\alpha} = \arg\min_\alpha \Delta_\alpha$, 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[4].

2.3 Illustration

We illustrate our framework on the well-known Zachary's Karate Club graph [6]. 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 $\hat{\alpha} = \arg\min_\alpha \Delta_\alpha$. In the right plot of Figure 1, we used UMAP (Uniform Manifold Approximation and Projection) [7] to 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.

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

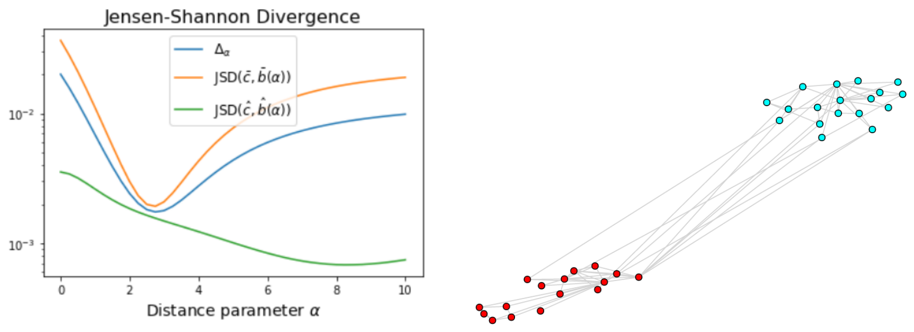

Fig. 1. Zachary's Karate Club Graph. We illustrate the divergence score as a function of α (left) for the best embedding found by our framework (right). The colors represent the two ground-truth communities.

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.

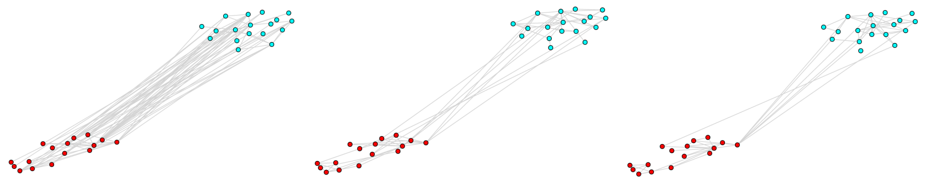

Fig. 2. Zachary's Karate Club Graph. We generate random edges following the Geometric Chung-Lu Model with the same expected degree distribution and with the highest scoring embedding. We look at three cases: $\alpha = 0$ which ignores the embedding (left), $\alpha = 7$ which penalizes long edges too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

3 Geometric Chung-Lu Model

It is known that classical Erdős-Rényi (binomial) random graphs $G(n, p)$ can be generalized to the Chung-Lu model $G(\mathbf{w})$, the random graph with a given expected degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$. It is a classic and well-known model but unfamiliar reader is directed to, for example, [1] or [8]. 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 models, that are called *spatial models* or *geometric graphs*, vertices are embedded in some metric space and link formation is influenced by the metric distance between vertices. 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.

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:

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

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

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

where

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

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, $g(d_{\min}) = 1$ and $g(d_{\max}) = 0$; in the computations, we can use clipping to force $g(d_{\min}) < 1$ and/or $g(d_{\max}) > 0$ if required. Let us also note that if $\alpha = 0$ (that is, $g(d) = 1$ for any $d \in [0, \infty)$ with $g(d_{\max}) = 0^0 = 1$), 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 $i \in [n]$. Let us mention one technical assumption. It might happen that $p_{i,j}$ is greater than one and so it should really be regarded as the expected number of edges between v_i and v_j; for example, as suggested in the book of Newman [2], one can introduce a Poisson-distributed number of edges with mean $p_{i,j}$ between each pair of vertices v_i, v_j.

In [1], we proved 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.

It is not clear how to find weights explicitly but they can be easily (and efficiently) approximated numerically to any desired precision, as is discussed in detail in in [1].

4 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 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 3 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 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. (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.) 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 $O(n \ln n)$.

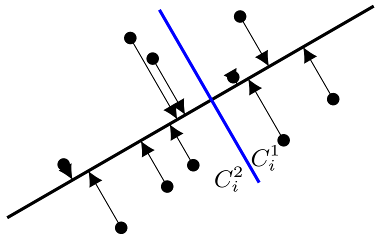

Fig. 3. Splitting C_i using SSE and the first principal component. Dots represent original points, thick black line represents the first principal component, and blue line represents the hyperplane orthogonal to the first principal component. It provides the desired split of C_i into C_i^1 and C_i^2.

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, $w'_i := \sum_{j \in C_i} w_j$.

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 across all strategies.

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 $i \in [n']$

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

Since it is an extension to [1], we revisited the proof of the uniqueness of weights in this more general setting (the proof is omitted here due to page limit). We showed that the weights exist and are unique if and only if the following condition is satisfied for more than two landmarks (for completeness, in the full version of the paper we also derived the conditions for the graph on $n' = 2$ landmarks, which are slightly different):

$$d_{t,t} > 0 \vee 2w'_t < \sum_{j \in n'} w'_j,$$

where $t = \arg\max_{j \in n'} w'_j$. 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 the College Football graph and testing the same set of embeddings as in [1]. This well-studied graph with known community structure represents the schedule of United States football games between Division IA colleges during the regular season in Fall 2000 [9]. The data consists of 115 teams (vertices) and 613 games (edges). 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 4 shows very high correlation between the two measures which indicates that the approximation algorithm preforms well, as expected.

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

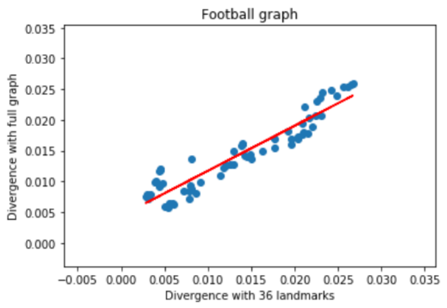

Fig. 4. College Football Graph exhibits high correlation between the original divergence score and its approximated counterpart.

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 5, we show the best and worst scoring embeddings for the approximated divergence score based on landmarks. The conclusion is the same as for the original algorithm: embeddings that score high are of good quality wheres the ones that score low are of poor quality.

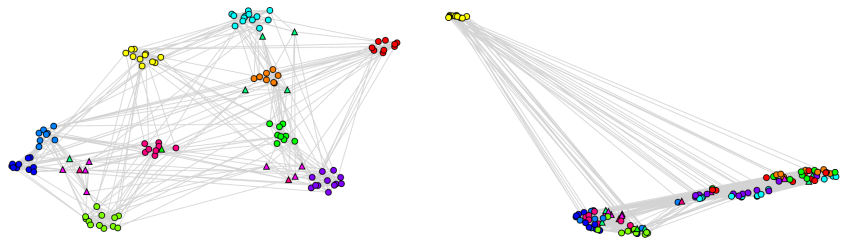

Fig. 5. The College Football Graph. We show the best (left) and the worst (right) scoring embedding based on the approximated algorithm with landmarks.

Our next experiment is with Email-Eu-core Network on 986 vertices. This network was generated using email data from a large European research institution and is available as one of the SNAP Datasets [10]. 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. We compare 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 6. As expected, there is a high correlation between the two sets with a satisfying outcomes already around $\sqrt{n}$ landmarks.

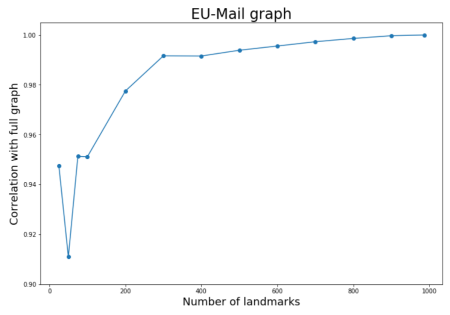

Fig. 6. Pearson's correlation between the divergence scores computed for the original graph on $n = 986$ vertices and the ones computed for the approximated variant on $n' \geq 25$ landmarks.

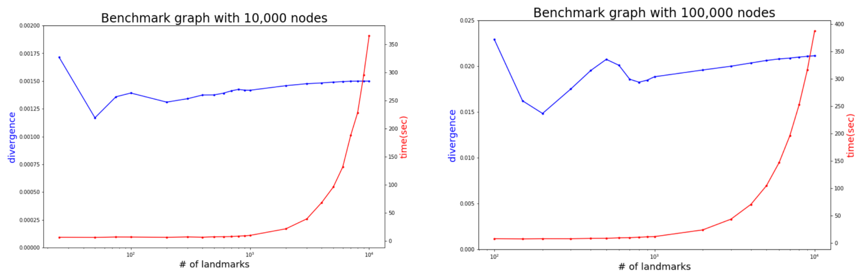

Fig. 7. Comparing quality and speed for ABCD graphs. We compare the approximated divergence score and the time required to compute it as a function of the number of landmarks.

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. The Artificial Benchmark for Community Detection (ABCD graph) [11] is a random graph model with community structure and power-law distribution for both degrees and community sizes (a new model that is an attempt to solve some of the problems of the standard method for generating artificial networks, the LFR graph generator [12]). Whereas the smaller graph can be easily tested by the original algorithm, dealing with the larger graph seems impractical (we only tested it for $n' \leq 10{,}000 < n = 100{,}000$ 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 7. 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 $n = (10^4)^2 = 10^8$.

5 Future Directions

In this paper, our aim was to introduce a scalable 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 generalizations of graphs in which a single (hyper)edge can connect any number of vertices. Hypergraphs are often more suitable and useful for representing and modelling many important networks and processes. We are interested in generalizing classic notions to hypergraphs, such as clustering via modularity [13], as well as developing new algorithms to apply to them [14]. 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. It 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. J. Janssen. Spatial Models for Virtual Networks. CiE 2010, LNCS 6158, pp. 201-210, 2010.
5. 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.
6. Zachary, W. W. An information flow model for conflict and fission in small groups. Journal of Anthropological Research 33, 452-473 (1977).
7. L. McInnes, J. Healy, J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. pre-print arXiv:1802.03426, 2018.
8. Chung FRK, Lu L. Complex Graphs and Networks. American Mathematical Society; 2006.
9. M. Girvan, M.E. Newman. Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99, 7821-7826 (2002).
10. J. Leskovec, A. Krevl. SNAP Datasets: Stanford Large Network Dataset Collection, http://snap.stanford.edu/data.
11. 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.
12. A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. Phys. Rev. E, 78(4), 2008.
13. B. Kaḿinski, V. Poulin, P. Prałat, P. Szufel, and F. Théberge, Clustering via Hypergraph Modularity, PLoS ONE 14(11): e0224307.
14. 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.