GRAPH IN MACHINE LEARNING
Continuity of Graph Embeddings

PATRICK SAUX
patrick.jr.saux@gmail.com

LE QUOC TUNG
quoctung.nus@gmail.com

SUPERVISOR
Philippe Preux
philippe.preux@inria.fr

Abstract

Graph is a complex structure which captures meaningful relations between entities and thus, enables many useful tools of Machine Learning in particular and Computer Science in general. Graph embedding is an approach which allows further exploitation of graphs potential. It aims to convert vertices, edges or even the graph itself into vectors, which are familiar forms to work with and can be used as input in standard machine learning tasks such as label prediction or clustering. Many graph embedding techniques have been introduced in literature. In this report, we aim to investigate a specific property of those techniques: the continuity. Theoretical and experimental results show that it is difficult to achieve continuity with current techniques, even in many simple graphs.

1 INTRODUCTION TO GRAPH EMBEDDING

Graphs appear naturally in many real life situations (e.g social media networks, citation graphs among researchers, knowledge graphs). The representation of graphs is crucial in many applications where the relations between entities encode useful and complex information. Graph analysis is a fundamental approach for many classical problems: node retrieving, node clustering, node classification and link prediction, to name a few.

However, many algorithms on graphs have high computation and storage cost. Working directly on graphs becomes prohibitive since traditional methods fail to scale with large amount of data. Therefore, the idea of graph embedding, which allows us to convert graphs (vertices, edges) into lower dimension vector spaces is very promising. With a good embedding, not only is the information of graphs structure preserved, but we can also enjoy working with vectors, which are much easier to manipulate.

In this project, the emphasis is on the continuity of graph embedding techniques. The question is whether a certain embedding technique defines a continuous transformation from the space of graphs to the embedding one. To be more specific, given two rather "similar" graphs, does the graph embedding technique produce two "close" representations? Studying this problem might greatly benefit many other applications which involve dynamic/evolving graph, real-time interaction, etc.

Adoption of graph embedding is heavily dependent on the type of input graph and desired output. There are various input graphs' type: *homogeneous graph*, *heterogeneous graph*, *graph with auxiliary information*, *graph constructed from non-relational data*, *directed graph*, *undirected graph*, *weight graph*, *non-weight graph*. Each type of graphs contains different information structures and it might resist the usability of certain embedding methods. On the other hand, depending on the purpose of the embedding, one would like *node embedding*, *edge embedding*, *hybrid embedding* and *whole-graph*

Preprint.

embedding. In this project, we heavily focused on the embedding of **homogeneous undirected weighted** graph into **node embedding** to have an in-depth analysis. Nevertheless, our analysis could be smoothly adapted to many other settings.

Following is the organization of the report: In the second section, the basic concepts and formal problem definition will be discussed. Section $3, 4, 5$ will be devoted for our attempt to introduce a set of graph embedding method, namely Eigenmap, Random Walk Matrix Factorization, which is an original method, and Deep Walk. After their formalisation, our focus is on the analysis of their continuity, followed by theoretical and experimental results. Section 7 presents experimental results on synthetic networks. In addition, two appendices cover additional material : a toy model to showcase the potential of biased random walk factorization, and a brief presentation of graph kernels for whole-graph embedding.

All methods have been implemented from scratch in Python (and Torch for the deep learning parts). Source code is available at : `https://github.com/sauxpa/graph_embedding`.

2 PROBLEM FORMALISATION

In this section, we will introduce basic definition of graphs and the formalisation of graph embedding problem. Most of definitions are borrowed from [3].

Definition 2.1 ***Graph***

A graph is $\mathcal{G} = (V, E)$, where $v \in V$ is a node and $e \in E$ is an edge. $\mathcal{G}$ is associated with a node type mapping function $f_v : V \to \mathcal{T}^v$ and an edge type mapping function $f_e : E \to \mathcal{T}^e$.

In the definition above, $\mathcal{T}^v$ and $\mathcal{T}^e$ are simply sets of vertex and edge type respectively.

Definition 2.2 ***Homogeneous graph***

A homogeneous graph $\mathcal{G} = (V, E)$ is a graph in which $|\mathcal{T}^v| = |\mathcal{T}^e| = 1$. All nodes in $\mathcal{G}$ belong to a single type and all edges belong to one single type. If either image set has more than one element, the graph is called heterogeneous.

Following are several criterias for graph embedding. Those are used to measure how well the embedding preserves the information of the graph.

Definition 2.3 ***k-th order proximity***

*The **first-order proximity** between node v_i and node v_j is the weight of the edge $e_{i,j}$. The **second-order proximity** between node v_i and node v_j is the similarity between v_i's neighborhood $s_i^{(1)}$ and v_j's neighborhood $s_j^{(1)}$ where $s_i^{(1)} = |e_{i,1}, e_{i,2}, \dots, e_{i,|V|}|$. Similarly, the **k-th order proximity** is defined recursively from the (k-1)-th order proximity.*

In short, first-order proximity is the local pairwise similarity between two neighbor nodes of the graph, the second-order proximity is the similarity between neighborhoods, etc...

Next, we introduce the formal definition of the graph embedding problem.

GRAPH EMBEDDING
Given a graph $\mathcal{G} = (V, E)$ and an integer d ($d \ll |V|$), the problem of graph embedding is to find a function $\mathcal{E}$ converting $\mathcal{G}$ into a d-dimensional space, in which the graph properties are preserved as much as possible. The graph properties can be quantified using proximity measures such as first or higher-order proximity. Each graph is represented as either a d-dimensional vector (for a whole graph) or a set of d-dimensional vectors with each vector representing the embedding of part of a graph (e.g, node, edge, substructure).

It is left only the definition of continuity of graph embedding. This requires to endow the space of graphs with a topology. Unfortunately, in certain settings, the natural topology for graphs is fundamentally discrete, which makes continuity either impossible or trivial. As we are most concerned with homogeneous undirected weighted graphs and the space of such graphs with a fixed

number of vertices N is homeomorphic to the Euclidean space of symmetric $N \times N$ matrices (via the adjacency matrix mapping), we adopt the following definition for node embedding continuity (with immediate extension to whole-graph, edge or other embeddings).

GRAPH EMBEDDING CONTINUITY
Let $\mathbb{G}_N$ the space of homogeneous undirected weighted graphs on N vertices, equipped with $\|\cdot\|_\infty$ the distance induced by the $\mathbb{L}^\infty$ norm on symmetric matrices and δ a distance on $\left(\mathbb{R}^d\right)^N$. The embedding map $\mathcal{E} : \mathbb{G}_N \to \mathbb{R}^d$ is δ-continuous if it is a continuous map between metric spaces $\left(\mathbb{G}_N, \|\cdot\|_\infty\right)$ and $\left(\left(\mathbb{R}^d\right)^N, \delta\right)$.

The choice of the distance δ on the embedding space allows for different notions of continuity. Since node embeddings correspond to clouds of N points in $\mathbb{R}^d$, we defined the following two metrics.

Definition 2.4 ***Embedding metrics***

Let $x, y \in \left(\mathbb{R}^d\right)^N$ two d-dimensional point clouds.

i $\delta_{\mathbb{L}^2}(x,y)^2 = \sum_{i=1}^N \|x_i - y_i\|_{2,\mathbb{R}}^2$ $(\mathbb{L}^2 distance)$,

ii $\delta_{W,2}(x,y)^2 = \min_{P \in Perm_N} \sum_{i,j=1}^N P_{ij}\|x_i - y_j\|^2$ *(2-Wasserstein)*

The $\mathbb{L}^2$ distance metricizes the $\mathbb{L}^2$ convergence of discrete measures supported by point clouds in $\mathbb{R}^d$ while the 2-Wasserstein distance metricizes the weak topology. Intuitively, $\mathbb{L}^2$ measures how far each point moves between two clouds (it is sensitive to permutation of the points), the other measure how far two clouds are when considered globally (invariant by permutation).

In contrast, if the topology of $\mathbb{G}_N$ or the embedding space is discrete, there is no clear, practical way to define continuity. This disadvantage is unavoidable since most graph settings involve discrete factors (for instance, neighbor graph is defined as one which has one more (or less) edge (or node)).

To deal with those cases, a rather simple methodology is proposed: we define (several) metric(s) to measure the distance between two embeddings. If the embedding between two neighbor graphs has small distance, that means the embedding is "continuous".

2.1 Continuity under edge removal

Let $\mathcal{G} = (V, E)$ and $e \in E$ an edge, we define $\mathcal{G} \setminus \{e\} = (V, E \setminus \{e\})$ the graph obtained by removing the edge e. We investigate the continuity of the embedding along the transformation from $\mathcal{G}$ to $\mathcal{G} \setminus \{e\}$. Note that, if we denote $e = (u, v)$, this corresponds to the continuity of this mapping along the direction supported by the elementary matrix $E^{uv} + E^{vu}$ ($E^{uv}_{ij} = \delta_{iu}\delta_{jv}$) corresponding to the graph with a unique edge between u and v; as a case of directional continuity, it is weaker than the continuity defined earlier on $\mathbb{G}_N$.

This setting covers several natural situations of evolving graphs. Firstly, edge removal and edge insertion are symmetric operations: studying the change of embedding of graph $\mathcal{G}$ and a one-edge-removed-neighbor $\mathcal{G}'$ is equivalent to that of graph $\mathcal{G}'$ and its one-edge-added-neighbor $\mathcal{G}$. Secondly, node insertion is a special case of edge insertion. We could assume that initially, graph $\mathcal{G} = (V \cup \{v\}, E)$ has an isolated vertex v. Adding a new vertex v into the graph is similar to adding an edge between v and another vertex in V. Interestingly, most matrix-based techniques extend naturally to augmented vertices set: for example the Laplacian matrices of $\mathcal{G}$ before and after adding v to V coincide on all but the last dimension, and so do their respective eigenvectors.

Definition 2.5 ***Edge removal loss***

We define the the following removal losses for embedding $\mathcal{E}$ with respect to embedding metric δ ($\mathbb{P}_E$ denotes a probability measure on E, usually the uniform distribution on edges):

i *e-removal loss:* $\delta\big(\mathcal{E}(\mathcal{G}), \mathcal{E}(\mathcal{G} \setminus \{e\})\big)$,

ii Max removal loss: $\max_{e \in E} \delta\big(\mathcal{E}(\mathcal{G}), \mathcal{E}(\mathcal{G} \setminus \{e\})\big)$,

iii Expected removal loss: $\mathbb{E}_{e \sim \mathbb{P}_E}\big[\delta\big(\mathcal{E}(\mathcal{G}), \mathcal{E}(\mathcal{G} \setminus \{e\})\big)\big]$.

Note that for large evolving networks, one can use the historical edge removal distribution to sample the last loss instead of calculating it for each possible edge.

With those definitions, we are ready for graph embedding techniques and their continuous property. Unless specified otherwise, we will consider the **WEIGHTED HOMOGENEOUS GRAPH** setting. In the next sections, we will successively introduce a few graph embedding techniques and discuss in-depth their continuity property.

3 Graph Enbedding with Eigenmap

3.1 Graph Laplacian Eigenmaps

Graph embedding with Laplacian Eigenmap is an instance of Manifold Learning ([2]) and is used for node embedding. Let us denote node embedding with Laplacian Eigenmap as $\mathcal{E}_{\text{eigen}}$.

The general idea of $\mathcal{E}_{\text{eigen}}$ is to preserve only local distances. In the context of a weighted graph, local distances are represented by $w_{i,j}$, which is the first-order proximity. For simplicity, we consider the graph embedding into $\mathbb{R}$. Let $y = (y_1, y_2, \ldots, y_{|V|})$ are the node embedding. The optimal embedding is the solution of the following optimization problem.

$$
\begin{aligned}
y^* &= \arg\min_y \sum_{i,j} w_{i,j} \left(y_i - y_j\right)^2 \\
&= \arg\min_y \sum_i y_i^2 \sum_j (w_{i,j}) - \sum_i \sum_j w_{i,j} y_i y_j \\
&= \arg\min_y y^T D y - y^T W y \\
&= \arg\min_y y^T L y
\end{aligned}
\tag{1}
$$

where D is the diagonal matrix whose element $d_{i,i} = \sum_j w_{i,j}$ and L is the Laplacian matrix of the graph. Since $\min y^T L y = 0$ and it is achievable, a normalization constraint is posed to prevent meaningless embedding where all vertices collapse to 0. Different types of constraint leads to different optimization problems (and solutions), a standard one is to constrain the norm of x to remove arbitrary scaling factor. The constraint is involved with diagonal matrix D. Since the bigger is the value $d_{i,i}$, the more important is vertex i. This kind of constraint implicitly contains the information of vertices. Following is its formulation:

$$
\begin{aligned}
\underset{x}{\text{minimize:}} \quad & y^T L y \\
\text{subject to:} \quad & y^T D y = 1
\end{aligned}
$$

The analytic solution for this problem is $y = f_2$ where f_2 is the second smallest eigenvector of the generalized eigenproblem $Lf = \lambda Df$.

Extension of this idea for graph embedding in $\mathbb{R}^d, |d| < V$, we consider from second smallest eigenvector to the $(d+1)^{th}$ one. Therefore, the embedding $\mathcal{E}_{\text{eigen}}(v_i) = \big(f_2(i), \ldots, f_{d+1}(i)\big)$.

Algorithm 1: Eigenmap embedding

Input Graph $\mathcal{G}$ embedding dimension d;
Output Graph embedding y;
$L \leftarrow Laplacian(\mathcal{G}), D \leftarrow Degree(\mathcal{G})$;
$(\lambda_i, E_{\lambda_i})_i$ eigenpairs of $D^{-1}L$ sorted in **ascending** order;
$(e^i_j)_j \leftarrow$ orthonormal basis of E_{λ_i};
$y = []$;
for $i \geq 2$ **do**
  for *j* **do**
    $y \xleftarrow{append} e^i_j$;
    if $dim(y) \geq d$ **then**
      break;
    end
  end
end

Note that if L admits a generalized eigenvalue of λ of multiplicity greater than 1, the eigenspace E_λ admits uncountably many orthonormal bases and the choice of the embedding vectors is then ill-defined. For simple eigenvalues, there are only two such bases ($\pm\frac{u}{\|u\|}$ for any nonzero eigenvector u), this choice is unique once an orientation is defined (for instance impose that the first coordinate of each vector is nonnegative).

3.2 The continuity of Eigenmap graph embedding

3.2.1 Continuity under weight perturbation

Since eigenmap embedding is constructed by (generalized) eigenvectors of the Laplacian matrix, it is sufficient to investigate the continuity of eigenvectors. This is a well-studied topic. In [1], eigenvalues and eigenvectors are continous functions if the eigenvalues are distinct. More precisely we have the following result:

Theorem 3.1 ***Eigenvectors continuity***

Let J a real interval, $t \in J \mapsto A(t)$ a continuous path in $\mathcal{S}_N(\mathbb{R})$. Assume that for all $t \in J$, $A(t)$ has distinct eigenvalues. Let $t \in J \mapsto \lambda(t)$ the (continuous) eigenvalue path, $t \in J \mapsto u(t)$ the corresponding eigenvector path such that $\forall t \in J, \|u(t)\| = 1$ and $\forall s, t \in J, u(t)^\top u(s) \geq 0$ (so that $u(t)$ is well defined as the unique unit vector of the eigenline associated with $\lambda(t)$ with a given orientation). Then u is a continuous path.

This is consistent with the earlier remark about multiple eigenvalues : defining the embedding in such cases is ill-posed, let alone the continuity of it.

As an immediate corollary, the eigenmap embedding $\mathcal{E}_{\text{eigen}}$ is continuous for all G such that the generalized eigenvalue problem $L(G)f = \lambda D(G)f$ has distinct solutions. It is interesting to discuss the problem of multiplicity of graph eigenvalues. Some of the most visible examples are graphs with $m \geq 2$ (strongly) connected components. In such cases, the multiplicity of eigenvalue zero is m.

Another example of eigenvalue multiplicity is the unweighted complete graph (or all elements of adjacency matrix is equal to 1). The solution for eigenvalue problem is:

$$\lambda_i = \begin{cases} 0 & \text{if } i = 1 \\ \frac{|V|}{|V|-1} & \text{otherwise} \end{cases}$$

Thus, for the unweighted complete graph, the multiplicity of eigenvalue $\frac{|V|}{|V|-1}$ is $|V|-1$. The space of eigenvectors of eigenvalue 0 and $\frac{|V|}{|V|-1}$ are $\mathbb{1}$ and $\{x|\mathbb{1}^T x = 0\}$ respectively. Note that in most applications, graphs with such a high level of symmetry are rare, especially weighted graphs.

4 Graph Embedding with DeepWalk

4.1 DeepWalk

As we can see in the previous section, $\mathcal{E}_{\text{eigen}}$ preserves the first order proximity. In this section, an embedding technique which preserves further orders of proximity will be introduced. It is called DeepWalk ([7]) and it makes use of Neural Networks (NNs) to train a presentation, which maximizes the probability of observing its neighborhood condition on their embedding. Deep Walk is inspired from Skipgram ([6]), a well-known word embedding technique in Natural Language Processing (NLP).

In this approach, graphs are treated as texts in NLP; vertices correspond to vocabulary and edges to possible syntactic and semantic relations; similarly, random walks on a graph (of fixed or varying size) are seen as sentences. The main idea of Skipgram is to train a vector representation of words in $\mathbb{R}^d$ in order to predict from a central word its context (surrounding words within a window of size w). Sampling sentences from the text and sliding the window w create batches of examples to train a NN and update its word representation via optimization algorithms (SGD and its variants).

The objective function for DeepWalk is:

$$\min_y \mathbb{P}(v_{i-w}, \dots, v_{i-1}, v_{i+1}, \dots, v_{i+w}|y_i) \tag{2}$$

Unfortunately, this objective function is too complicated to optimize. SkipGram makes use of a conditional independence assumption to simplify the problem:

$$\min_y \prod_{j=i-w, j\neq i}^{i+w} \mathbb{P}(v_j|y_i) \tag{3}$$

where $\mathbb{P}(v_j|y_i)$ is defined via a softmax as:

$$\min_y \mathbb{P}(v_j|y_i) = \frac{\exp y_j^T y_i}{\sum \exp y_j^T y_i} \tag{4}$$

Two tricks have been introduced in the literature to ease the training process: hierarchical softmax and **negative sampling**. Both of them propose a way to avoid calculating the normalisation term $\sum \exp y_j^T y_i$. In the latter we sample negative instances i.e pairs of word-context (w, c') where the c' is different from the actual surrounding words of w. Denote by $\mathcal{D}$ (resp. $\mathcal{D}'$) the set of positive (resp. negative) examples. The skipgram likelihood is then parametrized by a logistic function :

$$\mathcal{L}_{skipgram}(y, \mathcal{D}, \mathcal{D}') = \sum_{(w,c)\in\mathcal{D}} \log \sigma(y_w^\top y_c) + \sum_{(w,c')\in\mathcal{D}'} \log \sigma(-y_w^\top y_{c'}) \tag{5}$$

Producing fake samples $\mathcal{D}'$ is rather simple: for each node encountered in the random walk, draw a few nodes at random : these are highly unlikely to form an actual sentence of vertices. To improve the quality of these counterfeit context, NLP literature suggests to sample under a concave power law of the frequency of occurence; in our case, we chose to use an exponent of 0.75 and use the empirical frequency of nodes encountered in the successive walks.

Notice that two embeddings are simultaneoulsy learned : $w \mapsto y_w$ and $c \mapsto y_c$, the central word and context representation respectively. This is the strategy we use in the implementation.

Note that Eigenmap can also be interpreted as a random walk method since extracting the eigenspace associated with the lowest generalized eigenvalue of the graph Laplacian amounts to calculating an invariant measure for the standard random walk. Using this interpretation, it becomes clear why it is a first-order proximity method : calculating the invariant distribution for each node involves counting the flow exchanged with neighbor vertices. By contrast, DeepWalk does not approximate the asymptotic distribution after mixing but rather extract information from the transient behavior of the Markov walk.

Finally, note that random walks are very memory-efficient as they require to load only a small fraction of the graph each time, which explains the appeal of DeepWalk and related methods for large-scale graph mining compared to full matrix factorization.

4.2 Beyond DeepWalk: node2vec

Besides the skipgram technique borrowed from NLP, DeepWalk relies on a standard graph random walk, where transition probability are proportional to edge weights (the exact Markov transition matrix is $D^{-1}L$). One can consider other types of random walks to promote certain modes of graph exploration; this is the idea behind node2vec ([5]).

Let $p, q > 0$ two positive parameters and defines the weights:

$$\alpha_{pq}(x, y, z) = \begin{cases} \frac{1}{p} \text{ if } x = z, \\ 1 \text{ if } \{x, y, z\} \text{ is a triangle in } \mathcal{G}, \\ \frac{1}{q} \text{ otherwise.} \end{cases}$$

Let $(X_t)_{t\in\mathbb{N}}$ a stochastic process on V the set of vertices of $\mathcal{G}$ with transition probabilities defined as $\mathbb{P}\big(X_{t+1} = z | X_{t-1} = x, X_t = y\big) \propto w_{yz}\alpha_{pq}(x, y, z)$, where w_{yz} is the edge weight between vertices y and z (the proportionality coefficient is fixed by the constraint that probabilities sum to 1). X is called a **(p,q)-biased random walk**. For $p = q = 1$ it is equal to the standard random walk; for all other parameters, it is not a Markovian walk (since the transition kernel depends on the two prior states), however the joint process $\big(X_{t-1}, X_t\big)$ is Markovian.

Note that p controls the probability of revisiting previous nodes while q controls the probability of visiting nodes further away; setting $q < 1$ enforces a *depth-first sampling* behavior, $q > 1$ approximates a *breadth-first sampling*.

4.3 DeepWalk with time dynamic regularization

As far as we are aware, continuity results for DeepWalk and related embeddings techniques are not kown and are likely to involve more sophisticated tools than for eigenmap techhniques, if anything because knowing the steady state of the random walks is not sufficient so studying the mixing phase would be required. However the deep learning setting of DeepWalk allows for more flexibility, in particular through the definition of the loss function.

Let us consider a sequence of graphs $\mathcal{G}_1, \dots, \mathcal{G}_T \in \mathbb{G}_N$ (for simplicity we assume the vertices V are the same and only the edges change). Given the unstable nature of neural networks (sensitivity to weights initialization, non-convex loss...), even running DeepWalk multiple times on the same graph can lead to different embeddings. To mitigate this, one can train the DeepWalk network sequentially, starting from the weights learned to embed the previous graph (i.e $\mathcal{G}_{t-1}$ is a warm-up for $\mathcal{G}_t$). Intuitively this helps trap the network weights in the same local minimum of the loss function for all $\mathcal{G}_1, \dots, \mathcal{G}_T$.

A more rigorous approach is given in [4] and introduces the notion of time regularization. When minimizing the skipgram loss (5), one can add the constraint that y_t belongs to a ball of prescribed

radius around the previous embedding y_{t-1} for a given distance δ. The Lagrangian formulation of this problem yields the following algorithm:

Algorithm 2: Time-regularized DeepWalk

Input Graphs $\mathcal{G}_1, \dots, \mathcal{G}_T$, random walks parameters, regularization strength λ, embedding distance δ, embedding dimension d;

Output Graph embeddings $y_1, \dots, y_T$;

for $t = 1, \dots, T$ **do**
  for $v \in V$ **do**
    $\mathcal{D}_t, \mathcal{D}'_t \leftarrow RandomWalks(\mathcal{G}_t)$;
    $y_t \leftarrow \arg\min_{y \in \mathbb{R}^d} \mathcal{L}_{skipgram}(y, \mathcal{D}_t, \mathcal{D}'_t) + \lambda\delta(y, y_{t-1})$
  end
end

One key input is the embedding metric δ. As discussed, a more flexible notion of continuity can be encoded with the Wasserstein distance between point clouds. However, the minimization problem above is typically solved iteratively by backpropagation, which requires explicit derivatives of the loss function w.r.t network weights, which is not feasible with the Wassertein distance. Consequently, in practice we set $\delta = \delta_{\mathbb{L}^2}$.

5 Graph Embedding with Random Walk Matrix Factorization

Inspired from node2vec biased random walks, we designed a matrix factorization technique that incorporates elements of BFS vs DFS exploration. As far as we are aware, this is a new technique.

Recall the (p, q)-biased random walk $\big(X_t\big)$, one can form the Markov transition matrix of the 2-step process $\big(X_{t-1}, X_t\big)$. This is *a priori* a rather large matrix of size $N^2 \times N^2$ however it benefits from two sources of sparsity : (i) transition from (u, v) to (u', v') only happens if $v' = u$ and (ii) (u, v) is a feasible state if and only if it is an edge in $\mathcal{G}$. The former ensures there are at most N nonzero entries per row of the transition matrix, the latter means this matrix can be reduced to $2|E|$ dimensions (2 transitions per edge, one for each direction). This is of utmost importance for scaling: most large-scale applications deal with sparse matrices, for instance such that $|E| = \mathcal{O}\big(|V|\big)$, in which case the 2-step system scales asymptotically like the original network.

Denote by $\tilde{P}$ this transition kernel. Note that because of the asymmetry of the transition weights, $\tilde{P}$ is not symmetric. Under technical assumptions, $\tilde{P}$ admits an invariant distribution π characterized by $\tilde{\pi} = \tilde{P}\tilde{\pi}$ and converges to it (for connected graphs, random walks are typically ergodic). Very informally, $\tilde{\pi}_{(u,v)}$ describes the probability of asymptotic visit of state (u, v). In order to recover a distribution π on the vertices from the 2-step, one can marginalize over the first coordinate : $\pi_v = \sum_u \tilde{\pi}_{(u,v)}$. The resulting vector π yields a node embedding of $\mathcal{G}$, and we denote by $\mathcal{E}^{pq}_{rwf}$ the corresponding mapping. This construction extends naturally to higher dimensions using the same

marginalization procedure on the eigenvectors associated to the largest eigenvalues (the largest being 1 and corresponds to the invariant measure).

Algorithm 3: Randow Walk Matrix Factorization embedding

Input Graph $\mathcal{G}$ random walks parameters, embedding dimension d;
Output Graph embedding y;

$\tilde{P} \leftarrow$ 2-step random walk transition matrix;
$(\tilde{\lambda}_i, E_{\tilde{\lambda}_i})_i$ eigenpairs of $\tilde{P}^\top$ sorted in **descending** order;
$(\tilde{e}^i_j)_j \leftarrow$ orthonormal basis of $E_{\tilde{\lambda}_i}$;
$\tilde{y} = []$;
for $i \geq 2$ **do**
  for *j* **do**
    $\tilde{y} \xleftarrow{append} \tilde{e}^i_j$;
    if $dim(\tilde{y}) \geq d$ **then**
      break;
    end
  end
end
$y \leftarrow Marginal(\tilde{y})$

As discussed earlier, the Eigenmap embedding can be seen as an invariant measure calculation for the standard random walk. The (p, q)-RW Factorization transposes this idea to biased random walks, while circumventing the caveat of asymmetric walks. The relation between these techniques is summarized in Table 1.

	Asymptotic distribution	**Transient mixing**
Standard RW	Eigenmap	DeepWalk
(p,q)-biased RW	(p,q)-RW Factorization	node2vec

Table 1: Random walk-based techniques.

6 Experiments

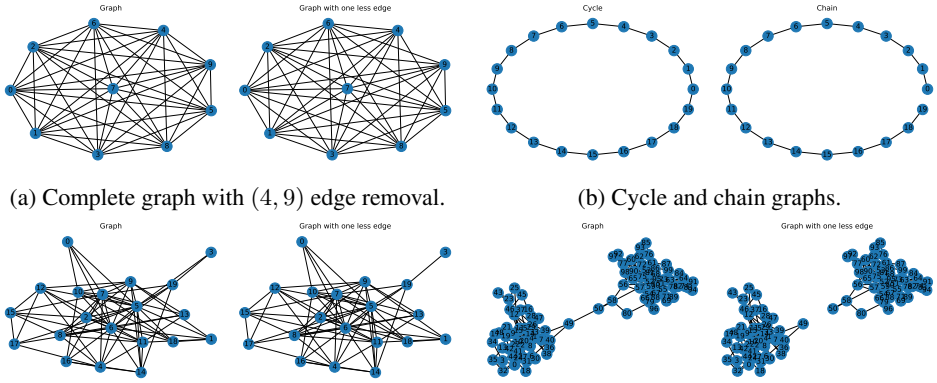
6.1 Settings

We propose a few simple graphs to study their embeddings and compare the effects of edge removal.

1. Complete unweighted graph (Figure 1a).
2. Cycle and chain graph (Figure 1b): In the first graph, we consider cycle graph $G = (V, E), E = \{(v_i, v_{i+1}), i = 1, 2, \dots |V|\}$ (with $v_{|V|+1} = v_1$). Weight is randomly sampled from the uniform distribution on $[0, 1]$.
3. Barabasi-Albert (BA) graph (Figure 1c) with parameter $|V| = 50, m = 5$. Weight is randomly sampled from the uniform distribution on $[0, 1]$. A random edge will be removed. Such graphs are classical instances of preferential attachment mechanism (random graphs built sequentially by adding edges with higher probability if a node is already connected to many other nodes) and are elementary models of social networks.
4. Two BA graphs with bridge (Figure 1d) with parameter $|V| = 50, m = 2$ for each BA graph. They are connected by only one bridge. Weight is randomly sampled from the uniform distribution on $[0, 1]$. The bridge will be removed.

6.2 Results

As predicted by the continuity result on eigenvectors, matrix-based embeddings vary drastically in the complete graph and BA bridge cases (see Figure 2a, 2d, 4a and 4d); by contrast DeepWalk seems



(a) Complete graph with $(4, 9)$ edge removal.

(b) Cycle and chain graphs.

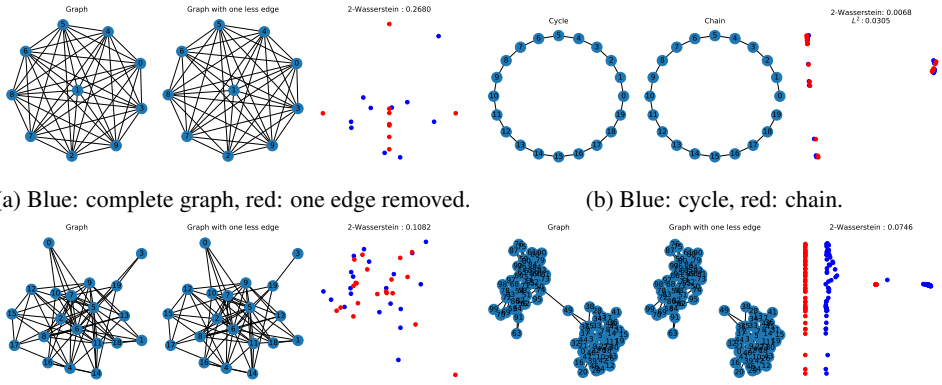
(c) Barabasi-Albert (BA) graphs with $(3, 5)$ edge removal.

(d) Barabasi-Albert (BA) communities linked by a single bridge.

Figure 1: Example of graphs.

to produce stable vector representations in both cases (Figure 3a and 3d). In the other examples, all tested methods produce reasonably close embeddings.

In the case of DeepWalk and its node2vec variants, we notice that different sensitivity to edge removal for BFS and DFS random walks.



(a) Blue: complete graph, red: one edge removed.

(b) Blue: cycle, red: chain.

(c) Blue: BA graph, red: one edge removed.

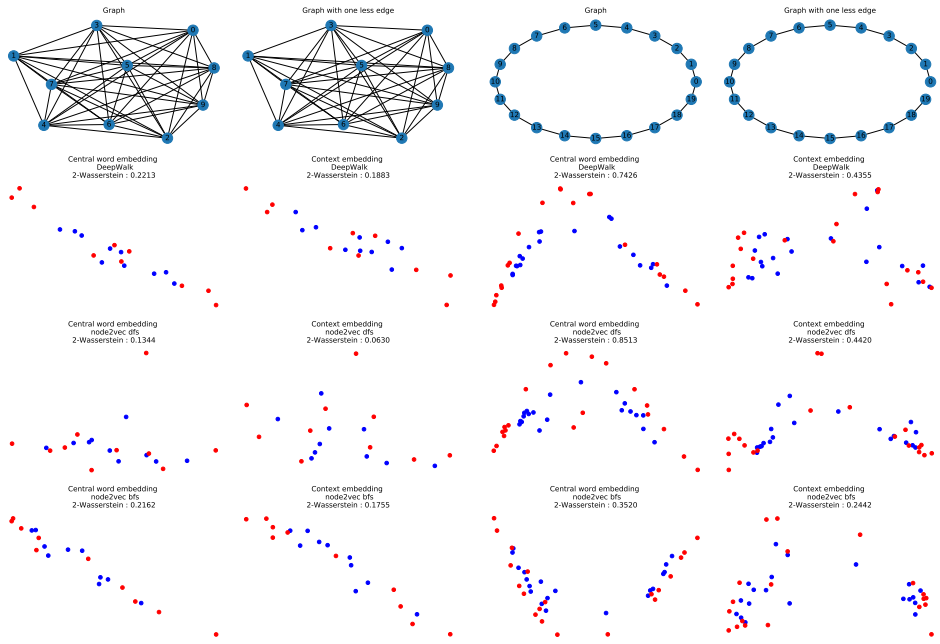
(d) Blue: BA bridge graph, red: bridge removed.

Figure 2: Eigennmap embedding.

Finally, the temporal regularization of DeepWalk is tested on the BA bridge removal experience. As expected, increasing the dynamic penalty results in smoother embedding evolution. Note that the distance between embeddings decreases in both Wasserstein and $\mathbb{L}^2$ metric even though the regularization is on $\mathbb{L}^2$: as argued, the latter is a more stringent metric, thus controlling the $\mathbb{L}^2$ smoothness also controls its Wasserstein counterpart.

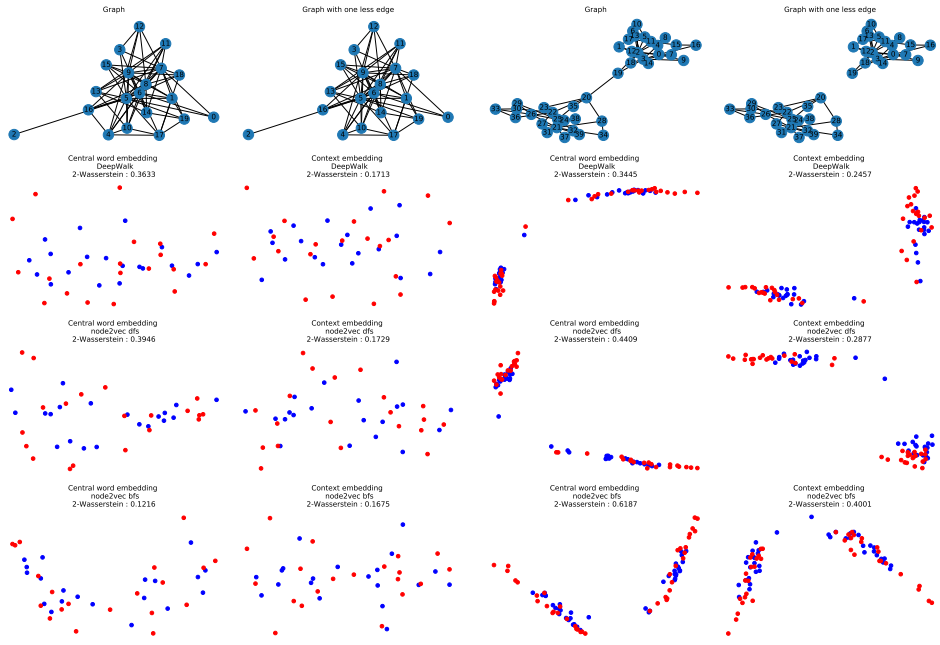
7 Conclusion

We have introduced a formal definition of embedding continuity for weighted homogeneous graphs as well as several metrics to assess embedding stability experimentally. For matrix factorization methods, theoretical guarantees of continuity exist as long as eigenvalues are distinct. In practice this is a common situation for connected graphs. Deep learning techniques based on random walks such DeepWalk and node2vec are less amenable to theoretical study, but can also be regularized to enforce sequential proximity in the loss function. Inspired by node2vec, we propose a variant of eigenmap based on the factorization of biased random walks. We conclude with an experimental framework to investigate continuity under elementary perturbations such as edge removal or reweighting.



(a) Blue: complete graph, red: one edge removed.

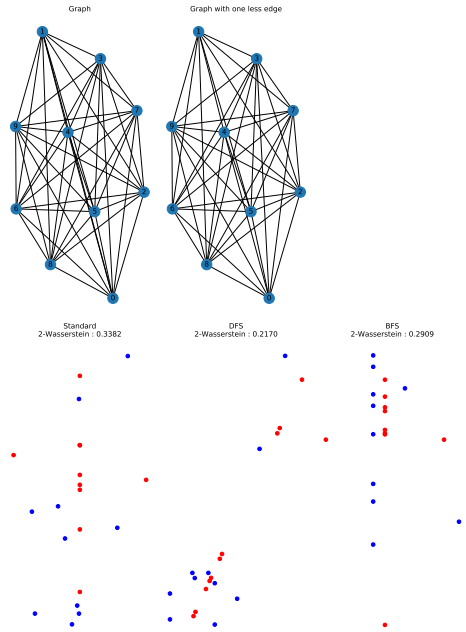
(b) Blue: cycle, red: chain.



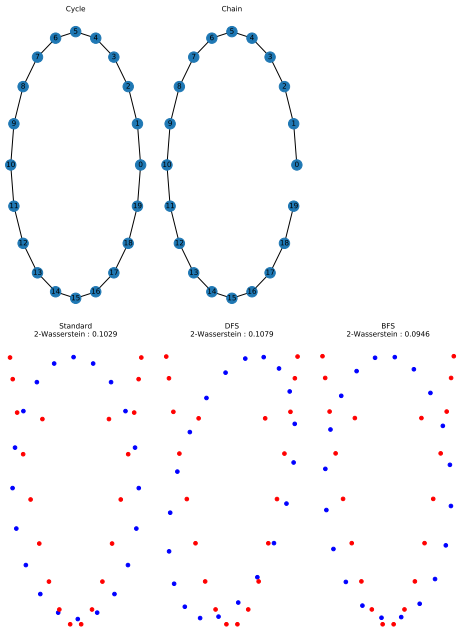
(c) Blue: BA graph, red: one edge removed.

(d) Blue: BA bridge graph, red: bridge removed.

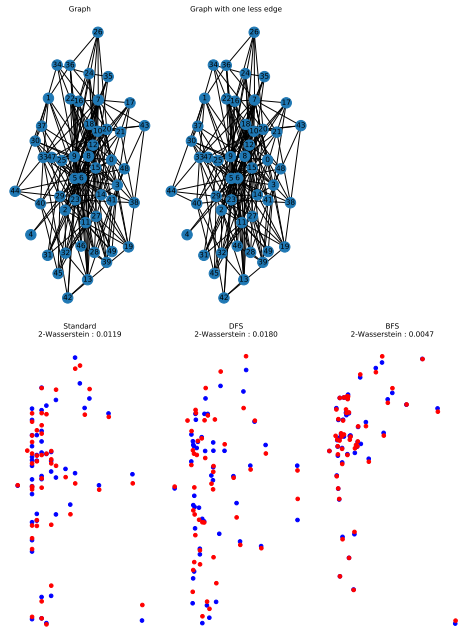
Figure 3: DeepWalk and node2vec embedding (dfs: $p = 1, q = 1/3$, bfs: $p = 1, q = 3$)



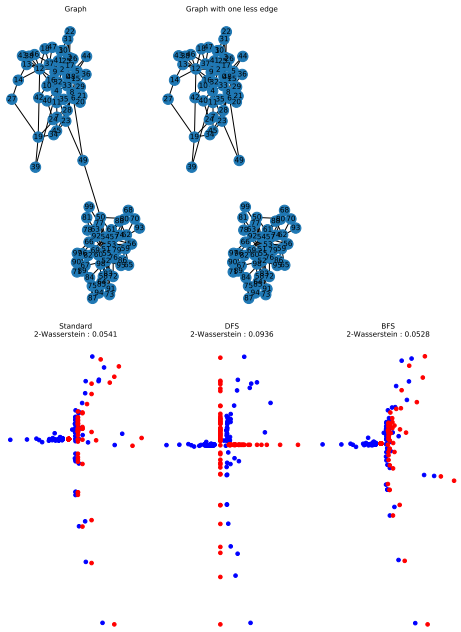
(a) Blue: complete graph, red: one edge removed.



(b) Blue: cycle, red: chain.



(c) Blue: BA graph, red: one edge removed.



(d) Blue: BA bridge graph, red: bridge removed.

Figure 4: Random Walk Matrix Factorization embedding (standard: $p = 1, q = 1$, dfs: $p = 1, q = 1/3$, bfs: $p = 1, q = 3$).

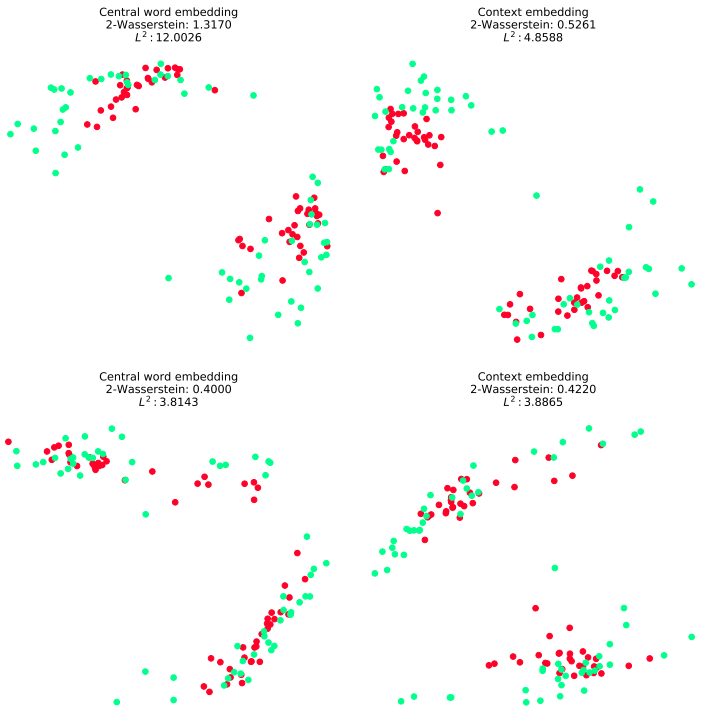


Figure 5: DeepWalk on BA bridge removal with temporal embedding regularization. Top: $\lambda = 0$ (no regularization). Bottom: $\lambda = 0.2$.

A Appendix: toy example of biased random walks

In this section, we investigate the sensitivity of the Random Walk Matrix Factorization embedding on a toy graph of 4 nodes with unweighted edges (Figure 6). This setting is simple enough to allow for analytic dependencies on parameters p and q and yet exhibits non-trivial embeddings. Note that other 4-nodes graphs with more symmetry (such as a chain, cycle or a star) have trivial invariant distribution for the (p, q)-biased walk, in the sense that it collapses to that of the standard walk.

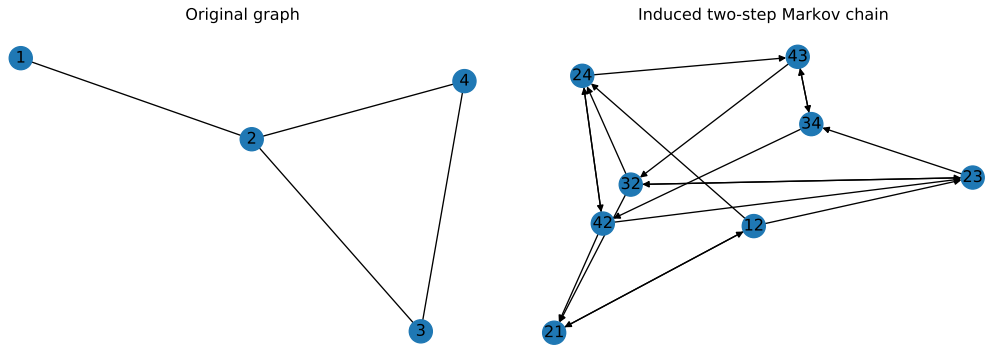


Figure 6

Let $\tilde{P}$ be the transition matrix of the induced 2-step Markov chain (see Figure 7 for numerical examples; explicit formula are an easy exercise of Markov chain calculus). It is of size 8×8 as there are 8 possible transitions in the original graph (two of opposite directions per edge). Computing the invariant distribution $\tilde{\pi}$ amount to solving the 8×8 system $\tilde{\pi}\tilde{P} = \tilde{\pi}$, which has the explicit solution $\tilde{\pi} \propto [\, 1, 1, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha \,]$ with $\alpha = \frac{p+q(p+1)}{2p+q}$. Finally we compute π by marginalization, i.e $\pi \propto [\, 1, 1 + 2\alpha, 2\alpha, 2\alpha \,]$.

To evaluate the impact of the random walk parameters, let us compute the distance ($\mathbb{L}^2$ or Wasserstein) between embeddings of the 4 nodes graph and each of the graph obtained by signle edge removal (see Figure 8). We see that BFS walks are more robust to the removal of the $1 - 2$ edge as it will favor the tightly knit cluster $2 - 3 - 4$; by contrast, DFS is less sensitive to the loss of $3 - 4$ as this link do not prevent in-depth exploration of the graph. The case of $2 - 3$ and $2 - 4$ (symmetric edges) is more balanced since this edge is both part of the cluster and leads to the central node 2.

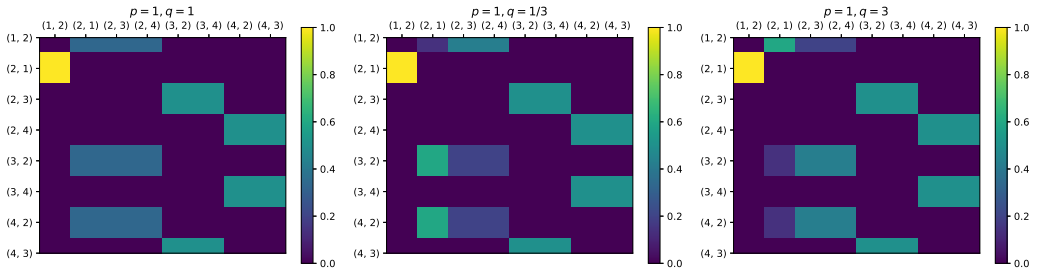


Figure 7: 2-step transition kernel for the toy 4 nodes graph embedding.

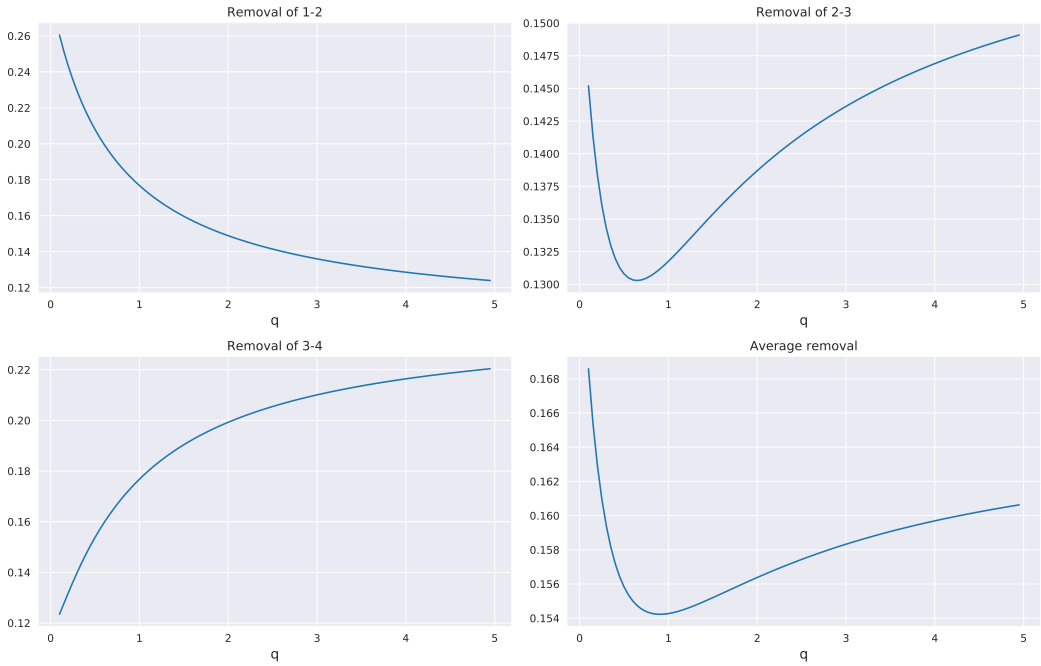


Figure 8: $\mathbb{L}^2$ edge removal loss and average loss for the 4 nodes toy graph.

B Graph Embedding with Graph Kernels

Graph Kernel ([8]) is a technique which provides embedding for the whole graph, not just vertices or edges. The general idea is to build a graph representation based on the count of atomic substructures or patterns. In our experiments, we consider two types of graph kernels: **Graphlet** and **Shortest path kernel**.

B.1 Graphlets

Graphlets are induced and non-isomorphic subgraphs of size-k. Suppose that the set of graphlets of size k is $\{G_1, G_2, \dots G_d\}$, then the graph could be embedded into vector v of d dimension where v_i is the number of induced subgraphs of G isomorphic with G_i. Figure 9 shows some instances of graphlets size 3.

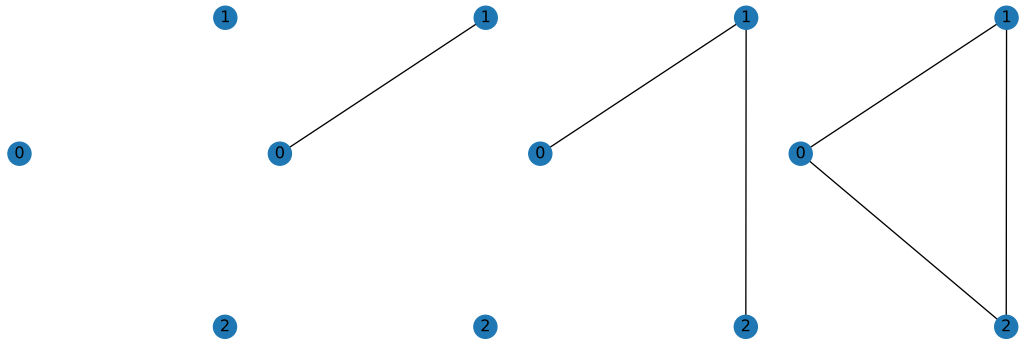


Figure 9: Isomorphism classes of graphlets of size 3

Trying all possible induced subgraph k is sometimes infeasible as it scales combinatorially with the size of the graph. Therefore, sampling uniformly induced k-subgraphs may be adopted to avoid heavy calculations.

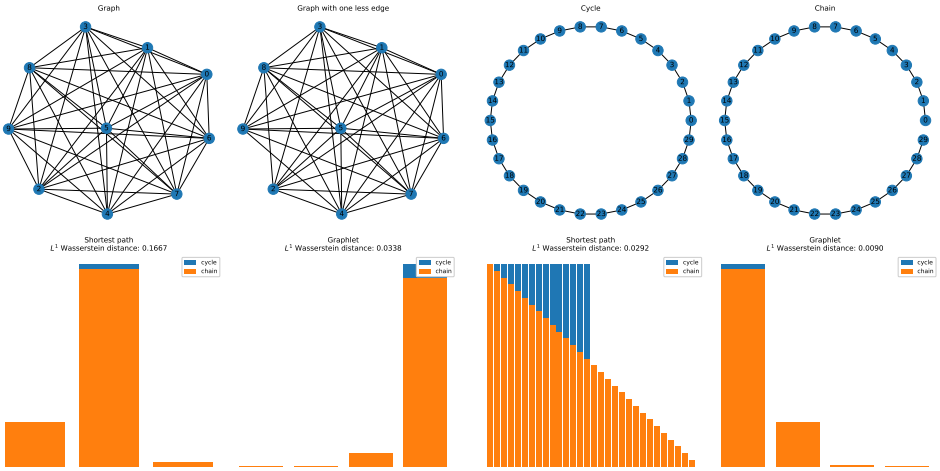
B.2 Shortest Path Kernel

Another kernel for graph embedding is shortest path kernel. The graph itself is transformed into a vector v of size $diam(\mathcal{G})$ (recall that the diameter of a graph is the longest of its shortest paths) where v_i is the number of shortest path having length equal to i.

B.3 Graph Kernels Continuity

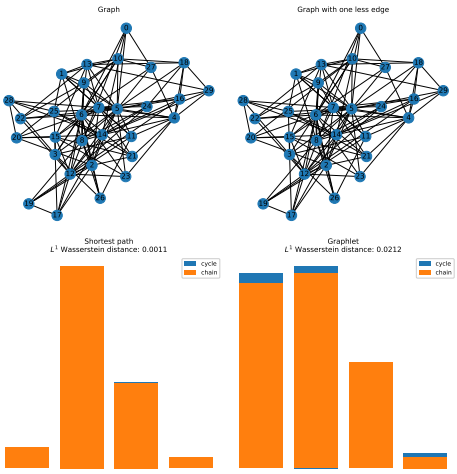
Graph kernel embeddings produce histograms of substructure occurrences. A natural topology on the space of histograms is, again, induced by the Wassserstein metric. In particular we have seen that shortest path kernel embedding dimension varies with the input graph, so a vector norm metric would be inappropriate, or at the cost of a cumbersome vector padding. However, the discrete nature of graph kernel embeddings makes it difficult to study formal continuity. In what follows, we discuss empirical observations based on the set of experiments defined in section 6.

While a more in-depth analysis would be required, continuity of a graph kernel embedding seems to depend on the local property of the kernel. Shortest paths are global features of a graph, and a small local change can dramatically alter the shortest path profile. In Figure 10b, a single edge removal doubles the diameter of the graph, making longer paths more likely; in Figure 10d, disconnecting the two communities removes the longest paths in the orignal graph, which correspond to inter-community paths, and each connected component is now treated as an independant graph. By contrast, graphlets are local structures which are not tremendously impacted by local perturbations. In Figure 10a, the complete graph only contains complete 3- graphlets; after removing one edge, it still mostly contains complete graphlets, only a tiny fraction of chain graphlets are formed. Similarly, graphlet histograms are visually close in all experiments, and the edge removal loss is generally lower than for shortest path kernel.

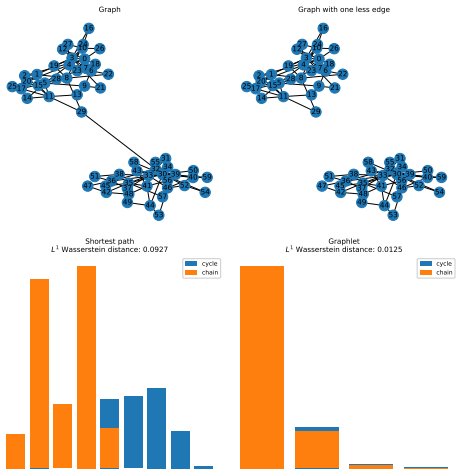


(a) complete graph.

(b) Cycle and chain.



(c) BA graph.



(d) BA bridge graph.

Figure 10: Graph kernel embedding.

References

[1] Andrew F. Acker. Absolute continuity of eigenvectors of time-varying operators. *Proceedings of the American Mathematical Society*, 42(1):198–201, 1974.

[2] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2003.

[3] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques and applications. *CoRR*, abs/1709.07604, 2017.

[4] Chuanchang Chen, Yubo Tao, and Hai Lin. Dynamic network embeddings for network evolution analysis, 2019.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[7] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014.

[8] S. V. N. Vishwanathan, Karsten M. Borgwardt, Imre Risi Kondor, and Nicol N. Schraudolph. Graph kernels, 2008.