Data Compression as a Comprehensive Framework for Graph Drawing and Representation Learning

Claudia Plant
Faculty of Computer Science, ds:UniVie
University of Vienna, Vienna, Austria
claudia.plant@univie.ac.at

Sonja Biedermann
Faculty of Computer Science
University of Vienna, Vienna, Austria
sonja.biedermann@univie.ac.at

Christian Böhm
Institute of Informatics, MCML
LMU Munich, Germany
boehm@ifi.lmu.de

ABSTRACT

Embedding a graph into feature space is a promising approach to understand its structure. Embedding into 2D or 3D space enables visualization; representation in higher-dimensional vector space (typically >100D) enables the application of data mining techniques. For the success of knowledge discovery it is essential that the distances between the embedded vertices truly reflect the structure of the graph. Our fundamental idea is to compress the adjacency matrix by predicting the existence of an edge from the Euclidean distance between the corresponding vertices in the embedding, and to use the achieved compression as a quality measure for the embedding. We call this quality measure Predictive Entropy (PE). PE uses a sigmoid function to define the probability which is monotonically decreasing with the Euclidean distance. We use this sigmoid probability to compress the adjacency matrix of the graph by an entropy coding. While PE could be used to assess the result of any graph drawing or representation learning method we particularly use it as objective function in our new method GEMPE (Graph Embedding by Minimizing the Predictive Entropy). We demonstrate in our experiments that GEMPE clearly outperforms comparison methods with respect to quality of the visual result, clustering and node-labeling accuracy on the discovered coordinates.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**.

KEYWORDS

Graph embedding; Minimum Description Length.

ACM Reference Format:
Claudia Plant, Sonja Biedermann, and Christian Böhm. 2020. Data Compression as a Comprehensive Framework for Graph Drawing and Representation Learning. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403174

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403174

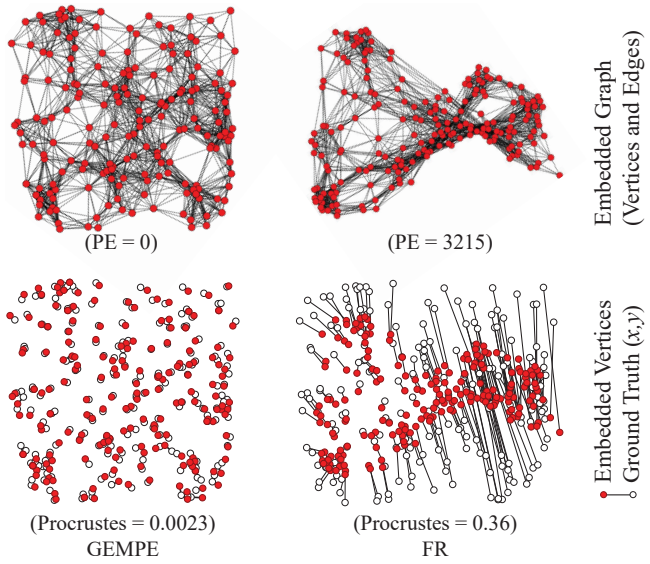

Figure 1: Comparing the Quality of Graph Drawings.

1 INTRODUCTION

What you see is what you get. Many applications naturally produce graph-structured data, e.g., social, traffic, or protein networks. A graph with n nodes is represented by a binary $n \times n$ adjacency matrix, or equivalently by an adjacency list. These representations are difficult to understand. It is hard to figure out the major patterns, even in very small graphs with only some tens of nodes. It is much more intuitive to view the graph in a 2D or 3D illustration with coordinates produced by a *graph drawing technique*.

Generating suitable vectors for the vertices of a graph is even more important in a second problem setting: Most techniques of machine learning and data mining like classification, clustering, anomaly detection, and regression work on vector data. Thousands of data analysis methods could be applied to graphs if we manage to translate graphs into vectors, typically in a medium to high dimensional space of ≥ 100 dimensions. This translation problem is referred to as *graph representation learning*.

How to map a graph to a low or high dimensional vector space? The research communities of graph drawing and representation learning have proposed sophisticated algorithms optimizing diverse goals (for more information see Section 5). Choosing the best technique to map a given graph is a difficult task. If we see some interesting structure in the embedding, we are convinced that this structure is present in the original graph. We prefer an embedding

where some pattern can be seen. However, the perceived structure can also be an artefact of the embedding technique. The top row of Figure 1 displays two drawings of the same graph: the result of our novel algorithm GEMPE on the left and the result of the widespread graph drawing algorithm by Fruchterman and Reingold (FR) algorithm on the right [8]. We might prefer the result of FR since it suggests that the graph has two sub-graphs which are connected by a dense bridge of nodes.

But is the interesting pattern found by FR really contained in the graph? The truth is revealed in the bottom-row diagrams which shows us how this artificial graph was generated: we first sampled 200 data points in 2D space from a uniform distribution (drawn as open circles). We then constructed a graph from this vector data set by connecting any pair of points (vertices) by an edge having a Euclidean distance less than a threshold ($\epsilon = 0.02$). By construction, this graph can be well mapped to 2D space—we even have a ground truth for the embedding in form of the originally sampled (x, y)-coordinates. In the bottom row of Figure 1 we show the ground truth coordinates (open circles) together with the embedding result (red points, connected to its ground truth point by a line), after globally rotating and aligning them to the ground truth by a technique called Procrustes analysis[1].

From the comparison between the learned coordinates and the ground truth we can draw two conclusions: (1) The "interesting" pattern of FR is a fake, merely produced as an artifact. The boring, uniform structure of GEMPE is very close to the true structure of generated data. And (2) we can also measure the error (called "Procrustes Error") of the reconstruction, which is considerable (0.36) in FR and in contrast very close to zero (0.0023) for GEMPE. It seems astonishing, that it is possible to reconstruct the original coordinates (of course, up to rotation, translation, etc.) at that high degree of precision from the binary information of the adjacency matrix only. How is this possible?

We consider graph embedding as a data compression task: The more effectively the low-dimensional coordinates allow to compress the information in the adjacency matrix, the better is the quality of the embedding. Any kind of structure or patterns in graphs including clusters, hubs and spokes is expressed in the adjacency matrix. By faithfully preserving the information in the adjacency matrix we expose any kind of interesting structure to visualization and further data mining steps.

Formal Problem Statement of Graph Embedding

We are given a graph $G = (V, E)$ with n vertices $V = \{1, ..., n\}$ and m (with $m \leq N = \frac{1}{2}n \cdot (n-1)$) unlabeled, unweighted, and undirected edges $(i, j) \in E$. Given is also the dimensionality $d \in \mathbb{N}$. The task is to find for each vertex $i \in V$ a vector $\mathbf{x}_i \in \mathbb{R}^d$ such that an objective function $f(\mathbf{x}_1, ..., \mathbf{x}_n)$ is minimized.

We define a novel objective function for graph embedding which we call the *Predictive Entropy* (*PE*). The *PE* function involves a probabilistic model predicting for any pair of vertices i and j the probability that an edge (i, j) exists. Given a good embedding, the *PE* function is able to predict all entries almost perfectly with probabilities close to zero for non-edges and close to one for edges. *PE* represents the number of bits required to encode the remaining uncertainty on the edge existence given the low-dimensional coordinates. A perfect embedding as in Fig. 1 has a *PE* of 0 bits.

[1] Alignment is necessary since any learning technique can reconstruct the original coordinates up to rotation, reflection, translation, and scaling.

Contributions

1. Based on the idea of compressing the adjacency matrix, we define Predictive Entropy as novel quality score for graph embedding (Section 2).
2. We introduce GEMPE (Graph Embedding by Predictive Entropy), an embedding technique optimizing *PE* (Section 3).
3. Extensive experiments demonstrate GEMPE's superiority for graph drawing and representation learning (Section 4).

2 PREDICTIVE ENTROPY

In this section, we develop an information-theoretic evaluation function $PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$ to assess the quality of an arbitrary embedding. We assume that each vertex i is already associated to a vector $\mathbf{x}_i \in \mathbb{R}^d$. Later, we will use this function to find an embedding that optimizes *PE*.

The general idea of most information-theoretic methods to data mining is to use the gained knowledge for data compression. If the data set can be efficiently compressed (with a small description length) with the help of the gained knowledge, then the knowledge represents a strong trend in the data set. For our problem definition at hand (graph embedding), the data set is the adjacency information, i.e. which vertex pairs are connected by an edge. The gained *knowledge* is the embedding, i.e. the vectors $\mathbf{x}_1, ..., \mathbf{x}_n$.

The basic representation of an undirected graph is the upper triangle of the adjacency matrix which requires a number $N = \frac{1}{2}n \cdot (n-1)$ of bits, one bit for each pair of vertices. This graph representation can be efficiently and lossless compressed by entropy-based methods like Huffman coding or arithmetic coding which assign to each object a bit string of the length $-\log_2 P$ where P is the (estimated) probability of the object. In a basic setting (without using our *knowledge*), we assume for each vertex pair the same probability of $P_{\text{basic}}\big((i, j) \in E\big) = \frac{m}{N}$ to be an edge where $m = |E|$ is the actual number of edges in our graph. Since we have a description length of $-\log_2 \frac{m}{N}$ for an edge and $-\log_2 \frac{N-m}{N}$ for a non-edge, the average description length of the vertex pairs corresponds to the basic entropy of the adjacency matrix:

$$\mathrm{H}_{\text{basic}} = -\frac{m}{N}\log_2\frac{m}{N} - \frac{N-m}{N}\log_2\frac{N-m}{N}, \tag{1}$$

which is between 0 and 1, often $\ll 1$. $\mathrm{H}_{\text{basic}}$ is also a lower bound of the size of the run-length coded adjacency matrix, the adjacency list, and compressed versions thereof.

However, with our knowledge (the embedding) we can get a better estimation of the probability that a vertex pair (i, j) is an edge, and thus a better compression. It is useful to apply the Euclidean distance $||\mathbf{x}_i - \mathbf{x}_j||$ between the associated vectors of the embedding and, instead of the constant probability $P_{\text{basic}}\big((i, j) \in E\big) = m/N$ to consider the conditional probability, given this distance:

$$P\big((i, j) \in E \,\big|\, ||\mathbf{x}_i - \mathbf{x}_j|| = \delta\big).$$

In our method *PE*, we model this conditional probability by a sigmoid function analogous to the (negative) cumulative normal distribution which is defined using the error function:

$$s^+_{\mu,\sigma}(\delta) := P\big((i,j) \in E \;\big|\; ||\mathbf{x}_i - \mathbf{x}_j||{=}\delta\big) = \tfrac{1}{2} - \tfrac{1}{2}\,\mathrm{erf}\big(\tfrac{\delta-\mu}{\sqrt{2}\sigma}\big),$$
$$s^-_{\mu,\sigma}(\delta) := P\big((i,j) \notin E \;\big|\; ||\mathbf{x}_i - \mathbf{x}_j||{=}\delta\big) = \tfrac{1}{2} + \tfrac{1}{2}\,\mathrm{erf}\big(\tfrac{\delta-\mu}{\sqrt{2}\sigma}\big).$$

Throughout this paper we will use superscripts to distinguish between the two cases *edge* ($^+$) and *non-edge* ($^-$).

The sigmoid function $s^+_{\mu,\sigma}(\delta)$ is depicted in Figure 2 (upper diagram). It combines several desirable properties:

1. It is strictly monotonically decreasing over the whole domain of definition. Thus, the optimization function applies a contractive force to all pairs of vertices which are connected by an edge and a repelling force for all pairs without edge.
2. It decreases from a value very close to one for very small distance values to a value very close to zero for large distances. For pairs of vertices having a very small or a very large distance, the number of required bits for the adjacency is close to zero.
3. The sigmoid function has two plateaus of small slope at either of its end, and a stronger slope in the middle. Therefore, non-edge vertex pairs which are already well separated in the embedding as well as edge-connected vertex pairs with a small distance in the embedding do not influence much the objective function. The objective function is naturally focussed on those pairs which still need improvement.
4. The sigmoid function has two intuitive parameters, μ and σ, which control the position and the strength of the slope.

Since $s^+_{\mu,\sigma}(\delta)$ models the probability that two vertices are connected by an edge, the *description length dl* (the number of bits required by the (i,j)-entry of the adjacency matrix in an entropy-based coding) corresponds to the negative binary logarithm of $s^+_{\mu,\sigma}(\delta)$ or $s^-_{\mu,\sigma}(\delta)$:

$$dl^+_{\mu,\sigma}(\delta) = -\log_2\big(\tfrac{1}{2} - \tfrac{1}{2}\,\mathrm{erf}(\tfrac{\delta-\mu}{\sqrt{2}\cdot\sigma})\big)$$
$$dl^-_{\mu,\sigma}(\delta) = -\log_2\big(\tfrac{1}{2} + \tfrac{1}{2}\,\mathrm{erf}(\tfrac{\delta-\mu}{\sqrt{2}\cdot\sigma})\big)$$
$$dl_{\mu,\sigma}(i,j) = \begin{cases} dl^+_{\mu,\sigma}(||\mathbf{x}_i - \mathbf{x}_j||) & \text{if } (i,j) \in E \\ dl^-_{\mu,\sigma}(||\mathbf{x}_i - \mathbf{x}_j||) & \text{otherwise.} \end{cases}$$

Intuitively, *dl* punishes an entry (i,j) of the adjacency matrix with a long bit string if (1) it is an edge but $||\mathbf{x}_i - \mathbf{x}_j||$ is close to or even (much) greater than μ, or (2) if $(i,j) \notin E$ but $||\mathbf{x}_i - \mathbf{x}_j||$ is close to or even (much) less than μ. In contrast, if we have a beautifully drawn graph with only edges having a distance $\ll \mu$ and non-edges $\gg \mu$ then *dl* awards us with a *dl* very close to 0.

For an undirected graph, the adjacency matrix is symmetric, and only the upper triangle is coded. The *predictive entropy PE* of the graph is the average number of bits needed for each entry of the adjacency matrix after compression:

$$PE_G(\mathbf{x}_1, ..., \mathbf{x}_n) = \tfrac{1}{N} \cdot \min_{\mu,\sigma} \sum_{1 \le i \le n} \sum_{i < j \le n} dl_{\mu,\sigma}(i,j). \qquad (2)$$

The function $PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$ can be used to assess the result quality of an arbitrary graph embedding method, because it measures how well we can predict an edge from the distance of the corresponding nodes in the embedding. The parameters μ and σ are optimized in order to minimize $PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$. If any sigmoid exists which allows on average over all node pairs a good prediction of the edge existence, we have a good embedding, no matter what the values of μ and σ actually are. In general, if σ is small compared to μ, then $dl_{\mu,\sigma}(\delta)$ is very decisive, which tends towards small coding cost (provided that there are not many prediction errors of vertex pairs $\in E$ having a distance $> \mu$ or vertex pairs $\notin E$ having a distance $< \mu$). We state a few properties of the predictive entropy:

Lemma 2.1. *$PE_G(...)$ is invariant to translation and rotation.*

Proof. The definition of $PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$ is solely based on the Euclidean distances between the associated vectors $\mathbf{x}_i$. As distances are invariant to translation and rotation, so is $PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$. □

Lemma 2.2. *$PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$ is invariant to uniform scaling, i.e. if all $\mathbf{x}_i$ are multiplied by a scalar α, $PE_G(...)$ remains constant.*

Proof. If all vectors are multiplied by a scalar value α, the distances are also multiplied and become $\alpha \cdot ||\mathbf{x}_i - \mathbf{x}_j||$. If in the original embedding μ and σ were used, then we can now apply $\alpha \cdot \mu$ and $\alpha \cdot \sigma$ to obtain for the case $(i,j) \in E$:

$$dl^+_{\alpha\cdot\mu,\alpha\cdot\sigma}(\alpha \cdot \delta) = -\log_2\big(\tfrac{1}{2} - \tfrac{1}{2}\,\mathrm{erf}(\tfrac{\alpha\delta-\alpha\mu}{\sqrt{2}\alpha\sigma})\big) =$$
$$= -\log_2\big(\tfrac{1}{2} - \tfrac{1}{2}\,\mathrm{erf}(\tfrac{\delta-\mu}{\sqrt{2}\sigma})\big) = dl^+_{\mu,\sigma}(\delta).$$

(analogously for the case $(i,j) \notin E$). Since this is true for any arbitrary (μ,σ), we know if (μ,σ) is optimal for $(\mathbf{x}_1, ..., \mathbf{x}_n)$ then $(\alpha \cdot \mu, \alpha \cdot \sigma)$ is optimal for the scaled data set $(\alpha \cdot \mathbf{x}_1, ..., \alpha \cdot \mathbf{x}_n)$, and it leads to exactly the same sum of cost. □

To assess the result of an arbitrary embedding method we optimize (μ,σ) simultaneously by steepest descent. We will use Lemma 2.2 in the next section to argue that μ can be set to an arbitrary value if *PE* is used as objective function and the optimization method of GEMPE allows the vector set to adapt to a given μ.

3 MINIMIZING PREDICTIVE ENTROPY

Methods related to multidimensional scaling (MDS) enable us to automatically determine the coordinates $\mathbf{x}_i$ for objects of which the matrix of all pairwise distances $d_{i,j}$ is known. MDS based on Eigenvalue decomposition requires the distance function to be a metric (symmetric, positive definite, and observing the triangle inequality), but more advanced MDS methods, in particular Weighted Majorization (WM) [3], [10] support non-metric distance functions. Moreover, WM allows us to apply a weight $w_{i,j}$ to each pair specifying to which extent the corresponding distance is taken into account in the overall objective function. The technique aims at finding low-dimensional coordinates $\mathbf{x}_i$ that minimize the squared difference between the distance in the input matrix $d_{i,j}$ and the Euclidean distance $||\mathbf{x}_i - \mathbf{x}_j||$ of the generated vectors:

$$wm(\mathbf{x}_1, ..., \mathbf{x}_n) = \sum_{1 \le i \le n} \sum_{i < j \le n} w_{i,j} \cdot (||\mathbf{x}_i - \mathbf{x}_j|| - d_{i,j})^2.$$

3.1 Parabolas for Weighted Majorization

Our coding function $dl_{\mu,\sigma}(i,j)$ does not directly provide a matrix of pairwise distances $d_{i,j}$. However, since $wm(\mathbf{x}_1, ..., \mathbf{x}_n)$ optimizes for a squared error (a parabolic function), we can derive from $dl_{\mu,\sigma}(i,j)$ a parabola $w_{i,j} \cdot (||\mathbf{x}_i - \mathbf{x}_j|| - d_{i,j})^2$ which tightly approximates the

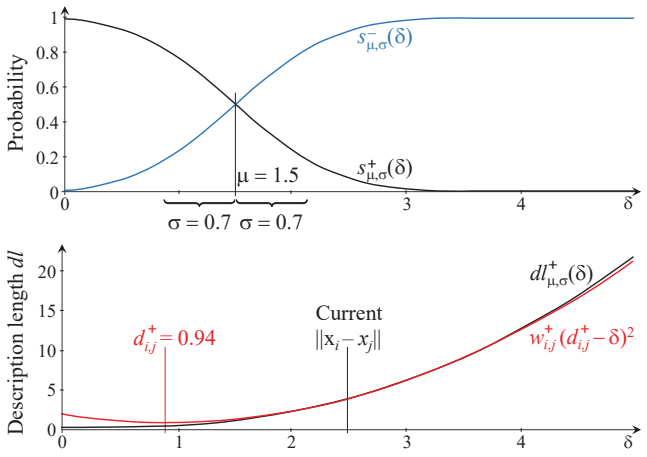

Figure 2: The Function $dl^+_{\mu,\sigma}(\delta)$ and its Approximation.

true cost (or, more precisely speaking, the increase or decrease of the true cost). We require that for the current distance $\delta = ||\mathbf{x}_i - \mathbf{x}_j||$, the *first and second derivative* of the description length function and the parabola agree. For $(i,j) \in E$ we obtain:

$$\frac{\mathrm{d}}{\mathrm{d}\delta} dl^+_{\mu,\sigma}(\delta) = \frac{\mathrm{d}}{\mathrm{d}\delta} w_{i,j} \cdot (\delta - d_{i,j})^2,$$
$$\frac{\mathrm{d}^2}{\mathrm{d}\delta^2} dl^+_{\mu,\sigma}(\delta) = \frac{\mathrm{d}^2}{\mathrm{d}\delta^2} w_{i,j} \cdot (\delta - d_{i,j})^2.$$

From these two equations, we determine $(d^+_{i,j}, w^+_{i,j})$ and obtain, after a few algebraic transformations:

$$w^+_{i,j} = \frac{\frac{2}{\pi \ln 2}\left(e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}\right)^2}{\sigma^2(1 - \mathrm{erf}(\frac{\delta-\mu}{\sqrt{2}\sigma}))^2} - \frac{\frac{1}{\sqrt{2\pi}\ln 2}(\delta-\mu)e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}}{\sigma^3(1 - \mathrm{erf}(\frac{\delta-\mu}{\sqrt{2}\sigma}))} \quad (3)$$

$$d^+_{i,j} = \delta - \frac{\frac{1}{\sqrt{2\pi}\ln 2} e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}}{w_{i,j}\sigma(1 - \mathrm{erf}(\frac{\delta_{i,j}-\mu}{\sqrt{2}\sigma}))}. \quad (4)$$

The corresponding formulas for the case $(i,j) \notin E$ are:

$$w^-_{i,j} = \frac{\frac{2}{\pi \ln 2}\left(e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}\right)^2}{\sigma^2(1 + \mathrm{erf}(\frac{\delta-\mu}{\sqrt{2}\sigma}))^2} + \frac{\frac{1}{\sqrt{2\pi}\ln 2}(\delta-\mu)e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}}{\sigma^3(1 + \mathrm{erf}(\frac{\delta-\mu}{\sqrt{2}\sigma}))} \quad (5)$$

$$d^-_{i,j} = \delta + \frac{\frac{1}{\sqrt{2\pi}\ln 2} \cdot e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}}{w_{i,j} \cdot \sigma \cdot (1 + \mathrm{erf}(\frac{\delta-\mu}{\sqrt{2}\sigma}))}. \quad (6)$$

and in the general case we write $(d_{i,j}, w_{i,j})$ with

$$(d_{i,j}, w_{i,j}) = \begin{cases} (d^+_{i,j}, w^+_{i,j}) & \text{if } (i,j) \in E \\ (d^-_{i,j}, w^-_{i,j}) & \text{otherwise.} \end{cases}$$

We can see this principle in Figure 2 where we have plotted the sigmoid function $s^+_{\mu=1.5,\sigma=0.7}(\delta)$ in the upper part and the corresponding cost function $dl^+_{1.5,0.7}(\delta)$ below. The parabola is determined such that at the marked position ($\delta = 2.5$), the two curves agree. We can see, that over a large range (approximately $1.5 \le \delta_{i,j} \le 3.5$) the actual *dl* is very well approximated by the parabola. Therefore,

algorithm GEMPE $(V, E, d) \to \mathbb{R}^{n \times d}$
  $\forall i \in V$: initialize $\mathbf{x}_i \in \mathbb{R}^d$ randomly (uniform distribution);
  repeat
    determine σ according to Section 3.2;
    $\forall i \in V$: $\mathbf{y}_i := (0, \ldots, 0)^\mathsf{T} \in \mathbb{R}^d$; // numerators and...
    $\forall i \in V$: $z_i := 0 \in \mathbb{R}$; // ...denominators of new points
(∗) **foreach** $(i,j) \in E$ **do**
      determine parabola $(d^+_{i,j}, w^+_{i,j})$ acc. to Eq. (3) and (4);
      weightedMajorizationStep (i,j);
      // negative sampling:
      sample $k \in V$ such that $k \neq i$ and $(k,i) \notin E$;
      determine parabola $(d^-_{k,i}, w^-_{k,i})$ acc. to Eq. (5) and (6);
      weightedMajorizationStep (k,i);
      sample $k \in V$ such that $k \neq j$ and $(k,j) \notin E$;
      determine parabola $(d^-_{k,j}, w^-_{k,j})$ acc. to Eq. (5) and (6);
      weightedMajorizationStep (k,j);
    $\forall i \in V$: $\mathbf{x}_i := \frac{1}{z_i} \cdot \mathbf{y}_i$;
  until convergence;
  return $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$;

procedure weightedMajorizationStep (a,b)
  // one step of weighted majorization for $\mathbf{x}_a$ and $\mathbf{x}_b$:
  $\mathbf{y}_a := \mathbf{y}_a + w_{a,b} \cdot \mathbf{x}_b$;
  $\mathbf{y}_b := \mathbf{y}_b + w_{a,b} \cdot \mathbf{x}_a$;
  $z_a := z_a + w_{a,b}$;
  $z_b := z_b + w_{a,b}$;
  if $||\mathbf{x}_a - \mathbf{x}_b|| \neq 0$ **then**
    $\mathbf{y}_a := \mathbf{y}_a + w_{a,b} \cdot \frac{d_{a,b}}{||\mathbf{x}_a - \mathbf{x}_b||} \cdot (\mathbf{x}_a - \mathbf{x}_b)$;
    $\mathbf{y}_b := \mathbf{y}_b + w_{a,b} \cdot \frac{d_{a,b}}{||\mathbf{x}_a - \mathbf{x}_b||} \cdot (\mathbf{x}_b - \mathbf{x}_a)$;

Figure 3: The Algorithm GEMPE.

we can move $\mathbf{x}_i$ and/or $\mathbf{x}_j$ over a wide area in the next iteration without leaving the area of good approximation. The parabola (with its parameters $d_{i,j}$=0.94, $w_{i,j}$=1.24) is used to determine the next set of coordinates.

As indicated before, we use Weighted Majorization [10] for the computation of the next set of vectors given the parameters of the parabolas $(d_{i,j}, w_{i,j})$. This iterative algorithm sets the new value $\mathbf{x}_i$ to:

$$\mathbf{x}_i := \frac{\sum_{j \neq i} w_{i,j} \cdot (\mathbf{x}_j + s_{i,j} \cdot (\mathbf{x}_i - \mathbf{x}_j))}{\sum_{j \neq i} w_{i,j}} \quad (7)$$

where $s_{i,j} = \begin{cases} d_{i,j}/||\mathbf{x}_i - \mathbf{x}_j|| & \text{if } ||\mathbf{x}_i - \mathbf{x}_j|| \neq 0 \\ 0 & \text{otherwise.} \end{cases}$

For a fixed sigmoid setting (μ, σ), we can provide the following algorithm: perform a loop until convergence which involves (1) determination of $(d_{i,j}, w_{i,j})$ for all pairs of nodes (i,j), and (2) determination of a complete set of new coordinates $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ for all nodes according to Eq. (7). The proof of convergence of WM can be found in [10]. In the next subsection, we provide a more general **linear** algorithm which also automatically selects the parameters (μ, σ) to minimize *PE*.

3.2 Mean and Variance of the Sigmoid

Due to the scale invariance of *PE* proven in Lemma 2.2 we can either put some constraint on the overall distribution of $\mathbf{x}_i$ or we are free to fix either μ or σ of our sigmoid. If we fix, for instance μ to a certain value (say 1.5), then our method involving determination of the parabolas and Weighted Majorization will strive towards an embedding in which edge-connected node pairs have a distance

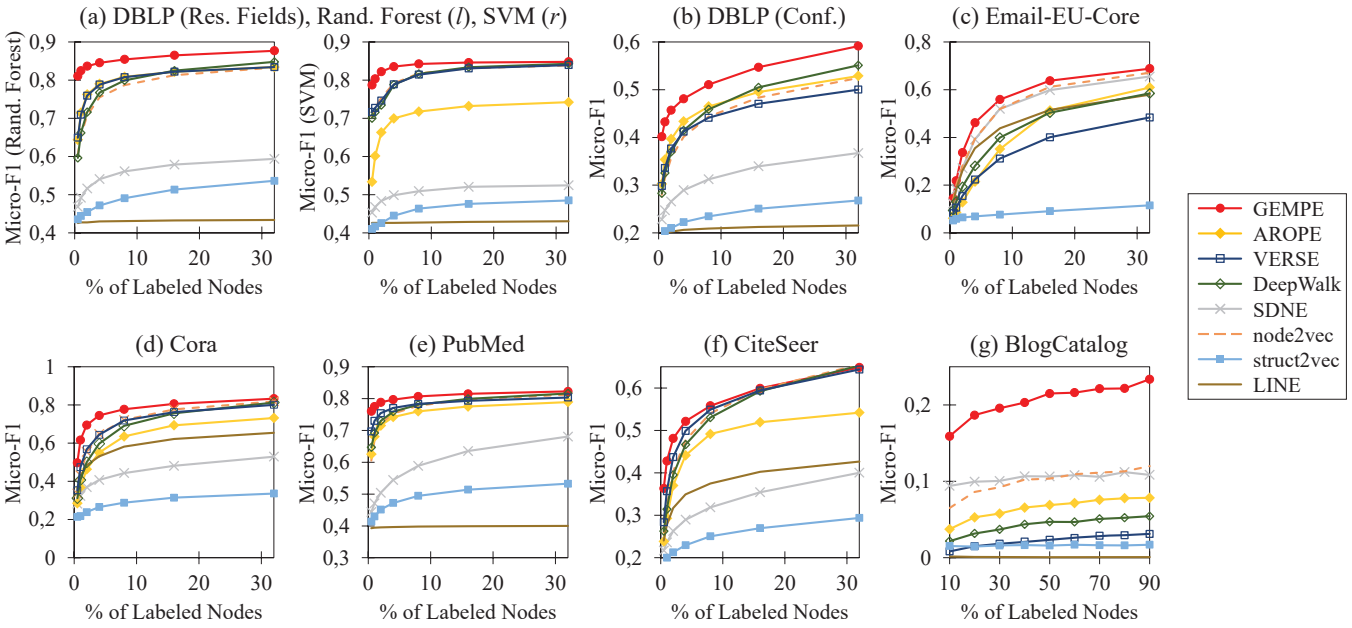

Figure 4: Comparison to Representation Learning Methods: Node Classification.

< 1.5, and non-edge pairs have a distance > 1.5, and depending on the goodness of our embedding, σ and our final cost will be small or big. Thus, it is very intuitive to fix μ since its meaning and its impact on the result quite obvious. We use a value between 1 and 2 because of the relationship to the path distance where edge-connected vertices have a distance of exactly 1 and not connected vertices have a distance ≥ 2.

If μ is fixed, then setting σ to minimize Eq. (2) involves a simple, one-dimensional search. We can easily determine the derivative $\frac{\mathrm{d}}{\mathrm{d}\sigma} PE_G(\mathbf{x}_1, ..., \mathbf{x}_n)$. It is easy to see that there is only one root which can be found efficiently by bisection search.

3.3 Algorithm GEMPE

The algorithm GEMPE is shown in Figure 3. After random initialization, GEMPE performs a loop until convergence which determines an optimal σ for the current embedding, and performs one iteration of the Weighted Majorization algorithm (starting with the (*)-marked foreach loop) where the optimal parabola is determined for each pair (i, j) of vertices, and the vectors associated to i and j, respectively, are then updated. In Eq. (7), the new points are defined as a fraction of which both numerator and denominator are determined from a sum, and thus it is most time efficient to collect numerator and denominator of all points separately (in the variables $\mathbf{y}_i \in \mathbb{R}^d$ and $\mathbf{z}_i \in \mathbb{R}$, respectively). Our algorithm is not considering all pairs of vertices, but performs negative sampling like [22], in the following way: we consider every edge $(i, j) \in E$ of the graph. While doing so, we also determine randomly a node $k \in V$ such that $i \neq k$ and there is no edge between i and k (edges are filtered out by a hash function). Consequently, we apply one step of weighted majorization (apply a repelling force between vertex i and k) for this pair of vertices. The same is done with a further sampled vertex which is not edge-connected to j.

Once the vectors $(\mathbf{x}_1, ...\mathbf{x}_n)$ are updated, the next round starts with the re-determination of σ. These alternating steps are repeated until convergence.

3.3.1 *Convergence and Complexity.*

Lemma 3.1. *GEMPE is guaranteed to converge after a finite number of steps.*

Proof. (1) Our objective function PE is bounded below by zero. (2) The objective function PE is monotonically decreasing in both steps, the weighted majorization step and in the (re-) determination of the sigmoid: (a) In the weighted majorization step, μ and σ of the sigmoid are fixed, and for this configuration, the positions of the embedded vertices are globally optimized. Since the positions of the vertices from the previous step are in the solution space of the current weighted majorization, the resulting cost cannot exceed those of the previous step. (b) In the re-determination of the sigmoid-parameter σ, the positions of the vertices in the embedding and thus their pairwise distances are fixed. σ is varied to optimize again the objective function PE, and since the previous σ is in the solution space, the predictive entropy cannot increase. Summarizing, the predictive entropy is monotonically decreasing, and (1) and (2) together guarantee convergence. □

The worst-case runtime complexity of an iteration is $O(|E| \cdot d)$, i.e. our algorithm is linear in the size (number of edges) of the graph. The worst-case space complexity of our algorithm is $O(|E| + n \cdot d)$. We need asymptotically no more memory than what is needed to store the input graph and its result.

3.3.2 *Weight Threshold.*

To increase the runtime efficiency we are interested in identifying those distances $\delta = ||\mathbf{x}_i - \mathbf{x}_j||$ for which the weight $w^-_{i,j}$ of non-edge pairs is below a threshold τ and can

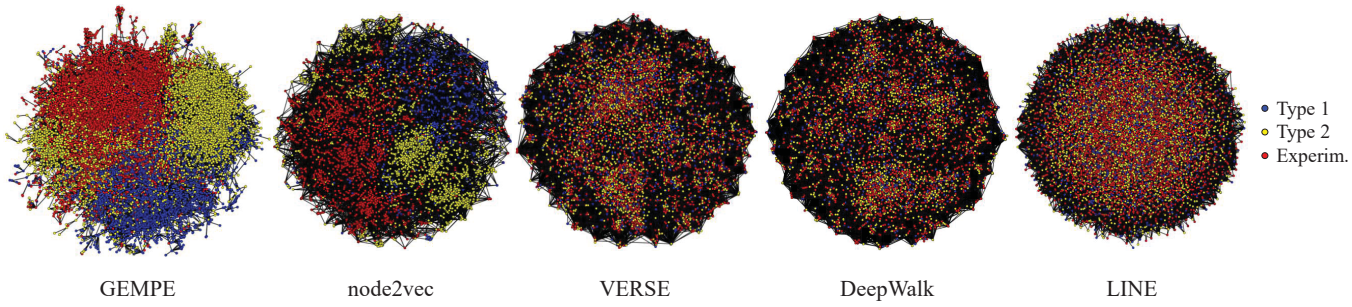

Figure 5: Visualization of PubMed Citation Graph

be neglected in Eq. (7). For $\delta \gg \mu$ the term $(1 + \text{erf}(\ldots)) \approx 2$ and $(\delta - \mu) \cdot \exp(\ldots) \gg \exp(\ldots)$, and therefore, we can approximate $w^-_{i,j}$ in the following way:

$$w^-_{i,j} \approx \frac{(\delta-\mu)e^{-\frac{(\delta-\mu)^2}{2\sigma^2}}}{2\sqrt{2\pi}\ln 2\sigma^3} \leq \tau \qquad \text{for } \delta \gg \mu.$$

This can be solved by the following iterative formula:

$$\xi := \sqrt{-\ln(4\pi\frac{\tau^2}{\xi^2}\ln(2)^2\sigma^4)} \quad \text{with initial } \xi := 1,$$

which converges fast (≤ 5 iterations) and the limit for $\delta_{i,j}$ where the weight is below τ is:

$$\delta \geq \sqrt{2} \cdot \sigma \cdot \xi + \mu.$$

Therefore, we store our set of vectors $\mathbf{x}_i$ using a d-dimensional grid where the distance of the grid lines in each dimension is $\sqrt{2}\sigma\xi + \mu$. Rather than considering all combinations (i, j) in Eq. (7), we consider only (1) all pairs $(i, j) \in E$ and (2) all pairs $(i, j) \notin E$ if they are in the same or adjacent grid cells, which improves the runtime. This is shown in Figure 6 where we see (in green) all considered partners of the red vertex: All vertices in the shaded neighboring grid cells plus all vertices with an edge to the red vertex. After a number of iterations where negative sampling is applied on the set of *all* non-edges, we focus on those non-edges in the same or adjacent grid cells.

3.3.3 Initialization. GEMPE can be arbitrarily initialized. We start with randomly selected coordinates for each vertex, taken from a uniform distribution.

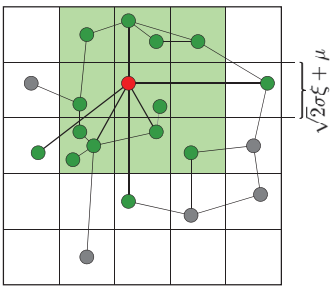

Figure 6: Vertex Storage in a Grid-Based Array.

4 EXPERIMENTS

We compare GEMPE to 10 state-of-the-art graph drawing techniques and 8 state-of-the-art representation learning techniques. Due to space limitations, we only can show a selection of the results. Please refer to the appendix for all necessary information to reproduce our experiments.

4.1 Representation Learning

To compare the quality of the vector space representation, we perform node classification experiments on graphs with node labels setting the dimensionality to 128, which is a common setting, see, e.g., [28]. We used random forests, neural networks, linear support vector machines and logistic regression as classifiers. All experiments are repeated 100 times and we report the averaged results. Figure 4 (a) shows the micro F1-measure on a graph extracted from DBLP using random forest and linear SVM. We extracted from DBLP all authors from the theoretical computer science conferences STOC, SODA and FOCS, at the database conferences SIGMOD, VLDB, ICDE and the data mining conferences KDD, ICDM and SDM in the years 2008 - 2017 (17,228 vertices, 161,354 edges). In Sub-figure (a) we consider the task to predict the research field of the author (3 labels). The coordinates of GEMPE provide the best prediction accuracy in all experiments. For random forest, already 0.5% corresponding to 86 labeled nodes allow to predict the label of the remaining 17,142 nodes with micro F1-measure of 84% which increases to 89% for 32% of labeled nodes. The runner-ups are AROPE [28] and VERSE [24] with F1-measure between 60 and 85%; As the classification results with different classifiers are similar and there are only minor differences between micro- and macro F1-measure, we only report random forest and micro F1 in the following. In Sub-figure (b) the task to predict the conference (9 labels). GEMPE achieves a F1-measure in the range of 40 and 60% which is superior to the accuracy of LINE [23], struct2vec [20] and SDNE [27] for the easier three-class problem. We also considered unsupervised clustering of the nodes into research fields and publication venues with K-means++, cf. Table 1. GEMPE is the best method with an NMI [26] of 0.38 followed by VERSE and DeepWalk [19] for k=3. For the difficult problem of clustering the venue, i.e. setting k to 9 we obtain similar results. Interestingly, the coordinates generated by AROPE are very suitable for classification but not at all for clustering. AROPE achieves only an NMI of 0.02 for k=3 and 0.09 for k=9. GEMPE outperforms the comparison methods also for node classification on the Email-EU-Core, CORA,

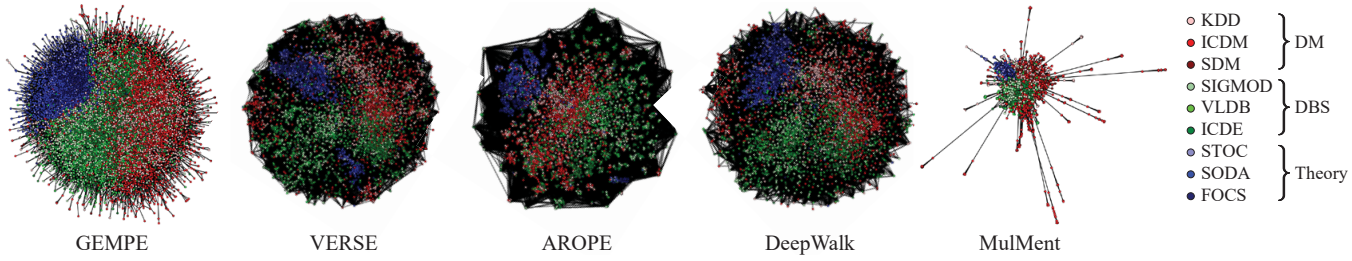

Figure 7: Visualization of DBLP Co-authorship Graph.

PubMed, CiteSeer and BlogCatalog networks. The best runner-ups are node2vec [14], VERSE and SDNE, cf. Figures 4 (c-g). On all these networks GEMPE performs especially well for small amounts of labeled nodes. Besides GEMPE, the methods VERSE, DeepWalk and node2vec support obtaining meaningful clusterings, consider e.g., the PubMed data consisting of 19,717 publications in three different areas of Diabetes research, and the 44,328 edges represent citations. We extracted 128D coordinates with all methods and performed K-means clustering setting K to 3 and a K-means++ initialization. The coordinates extracted by DeepWalk, GEMPE, and VERSE give good results.

4.2 Graph Drawing

Figure 7 shows the visualizations of the DBLP graph for GEMPE, graph representation learning techniques, and the graph drawing method MulMent [17]. The remaining graph drawing methods do not support the visualization of large networks. As it is common practice in representation learning literature, we use t-SNE [25] to map the 128D output coordinates of the comparison methods to 2D space. For GEMPE and MulMent, we directly obtain 2D coordinates, as it is common practice in graph drawing. GEMPE clearly separates the theoretical computer science from the two practical disciplines. Also MulMent achieves a good separation of the different research areas, which is, however, barely visible since most of the drawing area is occupied by a few nodes which are spiked out.

Figure 5 visualizes the PubMed graph. GEMPE achieves a good separation of all classes and also the fewest edge crossings. The result of VERSE much worse than that of GEMPE although the high-dimensional coordinates of VERSE perfectly contain the cluster information. This observation indicates that the cluster structure is to a large extent destroyed by the mapping to 2D space. The results of node2vec are better while LINE again performs worst. We could not compare to MulMent on this data set as the implementation produced NaN coordinates for this graph.

	DBLP *f.*	DBLP *c.*	Email	PubMed
GEMPE	**0.38**	**0.27**	0.67	0.29
AROPE	0.02	0.09	0.50	0.03
deepWalk	0.35	0.26	0.69	**0.30**
LINE	0.00	0.00	0.55	0.00
SDNE	0.02	0.04	0.62	0.02
node2vec	0.09	0.21	**0.70**	0.14
struct2vec	0.00	0.00	0.22	0.00
VERSE	0.36	0.25	0.67	0.28

Table 1: Comparison of Clustering Results (NMI) in 128D.

Figure 8 shows some exemplary drawings of the Minnesota road network together with their Predictive Entropy. Most specialized graph drawing methods produce good results with low PE (marked in darker color in the PE bar-graph). GEMPE (a) and the runner-up method FM3 [15] (b) provide a meaningful visualization without edge crossings. The graph drawing method LinLog [18] performs worse in PE than most other drawing methods and the 2D visualization is quite distorted, see Fig. 8 (c). The drawing of SDNE is typical for representation learning methods which tend to show a lower drawing quality and a lower PE, see Fig. 8 (d).

Figure 9 summarizes the graph drawing results on smaller labeled graphs: The Football data set consists of 115 nodes representing different college football teams and 613 edges representing the games played in the season 2000. The 12 classes are football conferences. GEMPE, DeepWalk and VERSE discover the cluster structure in this graph very well. LinLog does not use the 2D space and Multipole mixes some of the clusters. The Airflights data set consists of 322 nodes representing US airports (labeled according to the geographical location *Alaska, Hawaii, Overseas, etc.*), and edges representing flights. In all drawings, *Alaska* colored in blue is clearly separated. GEMPE, Multipole and DeepWalk also map *Hawaii* (cyan) and the overseas territories (green) to separated locations. We further observe that in the drawings of GEMPE and Multipole the US mainland is reasonable aligned with colors ranging from the east cost (red), Florida (magenta) over the midwest (orange) to the west coast (yellow).

Figure 10 shows the drawings of a finite element mesh which can be well mapped to 2D space. Similar as on the Minnesota road network, representation learning techniques tend to have difficulties. This is perhaps due to the approach of first mapping of an intrinsically 2D graph to 128D space and then back to 2D space with t-SNE. Both mapping steps can introduce errors which finally cause wrinkling and edge crossings. We also tried to parameterize representation learning methods to output 2D coordinates directly. However, the results are even worse.

5 RELATED WORK AND DISCUSSION

The task of finding a low-dimensional vector space representation for a graph has a long history and has been studied by different research communities. We categorize the variety of existing methods according to their different primary focus and the typical dimensionality of the extracted feature space. Methods from *information visualization* aim at drawing graphs, for a survey see [11]. Most related to GEMPE from the family of info viz techniques are distance scaling methods which aim at mapping the shortest path distances

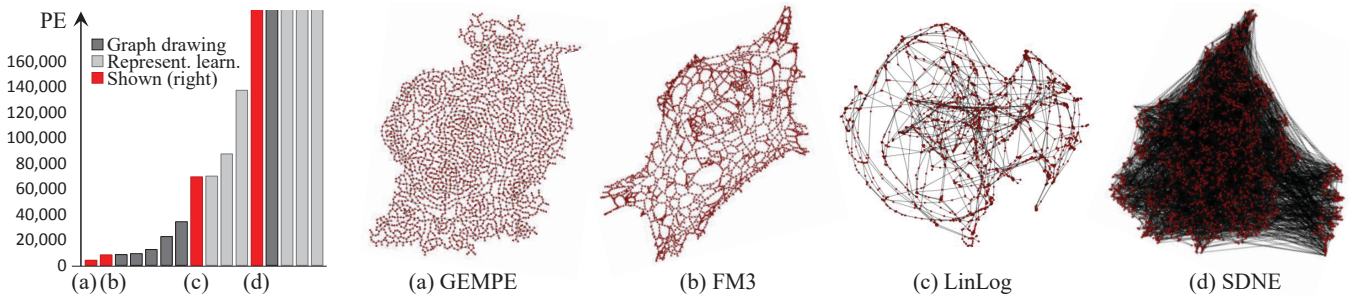

Figure 8: PE as a Quality Measure for Graph Drawing: Clear drawings of the Minnesota road network tend to have a low PE.

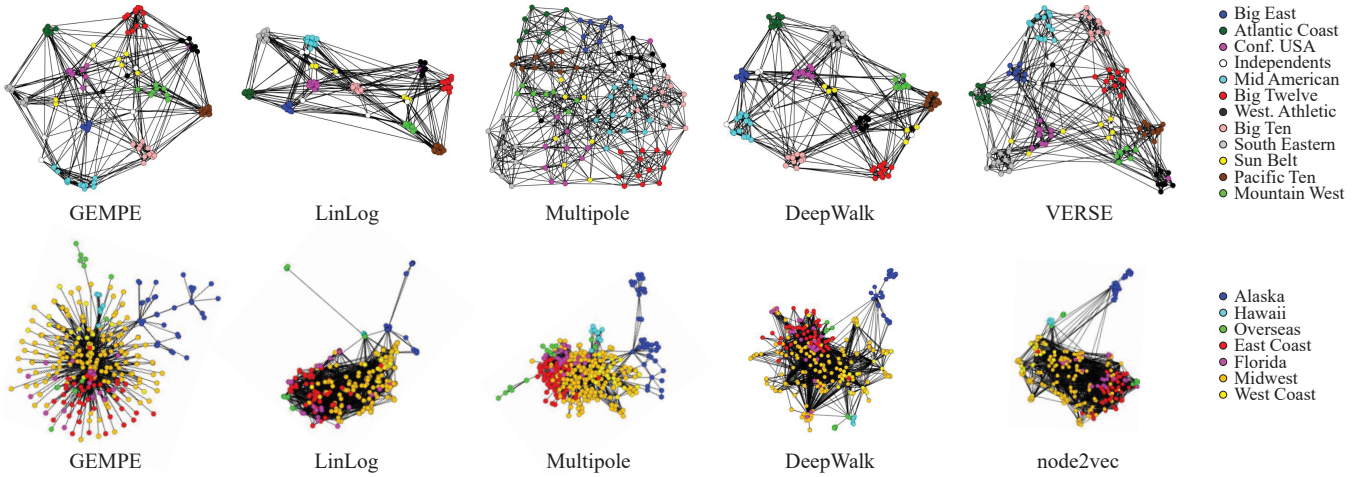

Figure 9: Drawings of Football and Airports Graph.

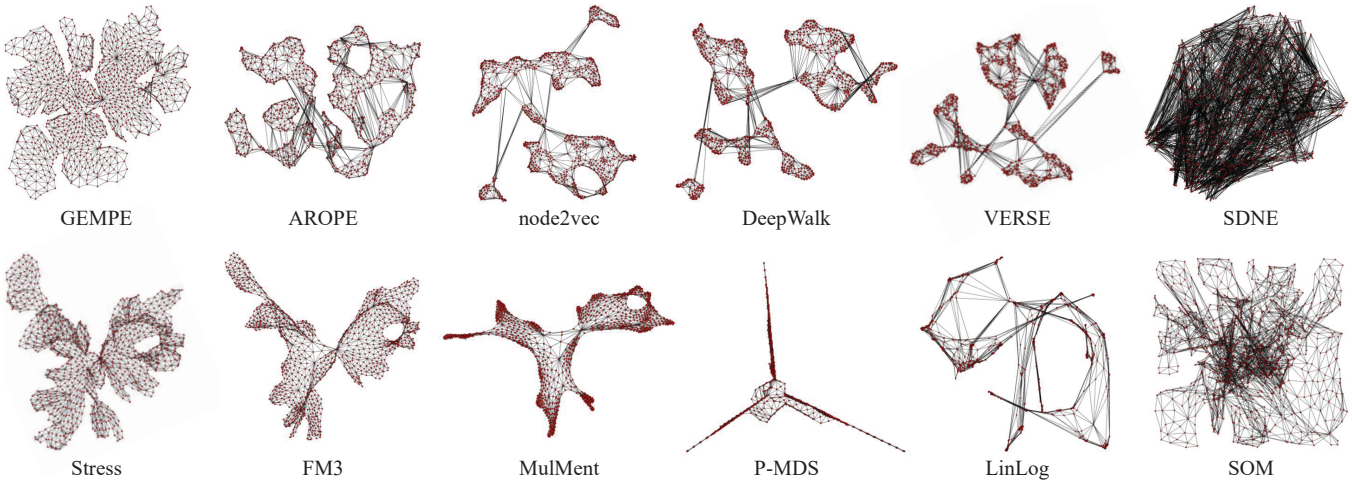

Figure 10: Drawings of Tapir Finite Element Mesh.

to 2D space, e.g., [6, 10, 17]. Most of these techniques are based on the statistical technique of Stress Majorization which supports weights for individual node pairs. Most commonly, a weight of $w_{i,j} = d_{i,j}^{-2}$ is applied, where $d_{i,j}$ corresponds to the shortest-path distance between node i and node j. Interestingly, this weighting is equivalent to the objective function of the classical Kamada Kawai algorithm. GEMPE exploits Majorization with a very different objective. Existing approaches consider distances and weights as input parameters. GEMPE iteratively learns target distances and weights optimizing the Predictive Entropy. We compare to MuMent [17] which is a recent multi-level stress-based method specially designed for drawing large graphs. In contrast to the previously mentioned methods, Multment supports graphs with several thousands of nodes as the DBLP data in Figure 7.

Recent *graph representation learning techniques*, for a survey see [16] focus on feature extraction from graphs. The methods DeepWalk [19] and node2vec [14] learn the embedding of a node based on random walks. These methods optimize a cross-entropy loss between the probability of a node pair to co-occur in a random walk and the Euclidean distance in vector space. The technique struc2vec [20] focuses on preserving structural roles of the nodes. In our experiments, DeepWalk and node2vec perform well in node classification and clustering. The technique LINE [23] considers preserving first-order and second-order node proximity, i.e. embeds nodes close to each other which are connected by an edge or have similar neighbors. Some recent methods use autoencoders to represent the local neighborhood of nodes, such as SDNE [27]. In our experiments LINE and SDNE have been outperformed by other methods, most importantly VERSE and AROPE. VERSE [24] is based on the idea that an expressive graph embedding should capture some similarity measure among nodes. As a default, the Personalized Pagerank similarity is preserved. VERSE produced very good coordinates for graph clustering. AROPE [28] is motivated by the fact the all other representation learning techniques can only preserve a fixed order of node proximity. An eigen-decompositon reweighting theorem supports to reveal node relationships across several orders of proximity. GEMPE preserves arbitrary order proximities by its self-adapting distance scaling strategy guided by the Predictive Entropy. We introduced related information-theoretic objective functions for other graph mining tasks, such clustering [12], summarization [5], and link prediction [4].

6 CONCLUSION

In this paper, we have introduced Predictive Entropy as a novel quality measure for graph embedding, as well as GEMPE (Graph Embedding by Minimizing PE), a graph embedding algorithm which uses PE as objective function. The basic idea is to relate graph embedding to the task of compressing the adjacency matrix by using the distances between embedded vertices as predictors for the edge existence. GEMPE preserves most information in the adjacency matrix in low-dimensional space as demonstrated by high clustering and node-labeling accuracy on real data. The PE score measuring the compression rate of the adjacency matrix also agrees with the visual quality of the result. Inspired by information theory and data compression our method does not require the user to specify any input parameters which are difficult to set. Features of any desired dimensionality can be extracted in a natural way. Our experiments demonstrate that GEMPE clearly outperforms comparison methods with respect to quality of the visual result, but also for data mining tasks like clustering and classification of nodes.

REFERENCES

[1] Christian Böhm, Martin Perdacher, and Claudia Plant. 2017. Multi-core K-means. In *SIAM Int. Conf. Data Mining*. 273–281.

[2] Ulrik Brandes and Christian Pich. 2006. Eigensolver Methods for Progressive Multidimensional Scaling of Large Data. In *Int. Symp. on Graph Drawing*. 42–53.

[3] Ulrik Brandes and Christian Pich. 2008. An Experimental Study on Distance-Based Graph Drawing. In *Int. Symp. on Graph Drawing*. 218–229.

[4] Jing Feng, Xiao He, Nina Hubig, Christian Böhm, and Claudia Plant. 2013. Compression-Based Graph Mining Exploiting Structure Primitives. In *ICDM*. 181–190.

[5] Jing Feng, Xiao He, Bettina Konte, Christian Böhm, and Claudia Plant. 2012. Summarization-based mining bipartite graphs. In *KDD conference*. 1249–1257.

[6] Linton C. Freeman. 2004. Graphic Techniques for Exploring Social Network Data. In *in Models and Methods in Social Network Analysis*. Univ Press.

[7] Arne Frick, Andreas Ludwig, and Heiko Mehldau. 1994. A Fast Adaptive Layout Algorithm for Undirected Graphs. In *DIMACS Graph Drawing Worksh.* 388–403.

[8] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph Drawing by Force-directed Placement. *Softw., Pract. Exper.* 21, 11 (1991), 1129–1164.

[9] Pawel Gajer and Stephen G. Kobourov. 2002. GRIP: Graph Drawing with Intelligent Placement. *J. Graph Algorithms Appl.* 6, 3 (2002), 203–224.

[10] Emden R. Gansner, Yehuda Koren, and Stephen C. North. 2004. Graph Drawing by Stress Majorization. In *Int. Symp. on Graph Drawing*. 239–250.

[11] Helen Gibson, Joe Faith, and Paul Vickers. 2013. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization* 12, 3-4 (2013), 324–357.

[12] Sebastian Goebl, Annika Tonch, Christian Böhm, and Claudia Plant. 2016. MeGS: Partitioning Meaningful Subgraph Structures Using Minimum Description Length. In *ICDM*. 889–894.

[13] Martin Gronemann. 2009. *Engineering the Fast-Multipole-Multilevel Method for multicore and SIMD architectures*. Master's thesis. Technische Univ. Dortmund.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.

[15] Stefan Hachul and Michael Jünger. 2004. Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm. In *Int. Symp. on Graph Drawing*. 285–295.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74.

[17] Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. 2018. Drawing Large Graphs by Multilevel Maxent-Stress Optimization. *IEEE Trans. Vis. Comput. Graph.* 24, 5 (2018), 1814–1827.

[18] Andreas Noack. 2007. Energy Models for Graph Clustering. *J. Graph Algorithms Appl.* 11, 2 (2007), 453–480.

[19] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD Conference*. 701–710.

[20] Leonardo Filipe Rodrigues Ribeiro, Pedro H. P. Saverese, and Daniel R. Figueiredo. 2017. *struc2vec*: Learning Node Representations from Structural Identity. In *KDD*. 385–394.

[21] Blake Shaw and Tony Jebara. 2009. Structure Preserving Embedding. In *ICML Conference*. 937–944.

[22] Noah A. Smith and Jason Eisner. 2005. Contrastive Estimation: Training Log-Linear Models on Unlabeled Data. In *ACL Conf.* 354–362.

[23] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW Conf.* 1067–1077.

[24] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW Conf.* 539–548.

[25] Laurens van der Maaten and Geoffrey E. Hinton. 2008. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[26] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *Journal of Machine Learning Research* 11 (2010), 2837–2854.

[27] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *KDD*. 1225–1234.

[28] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *KDD*. 2778–2786.

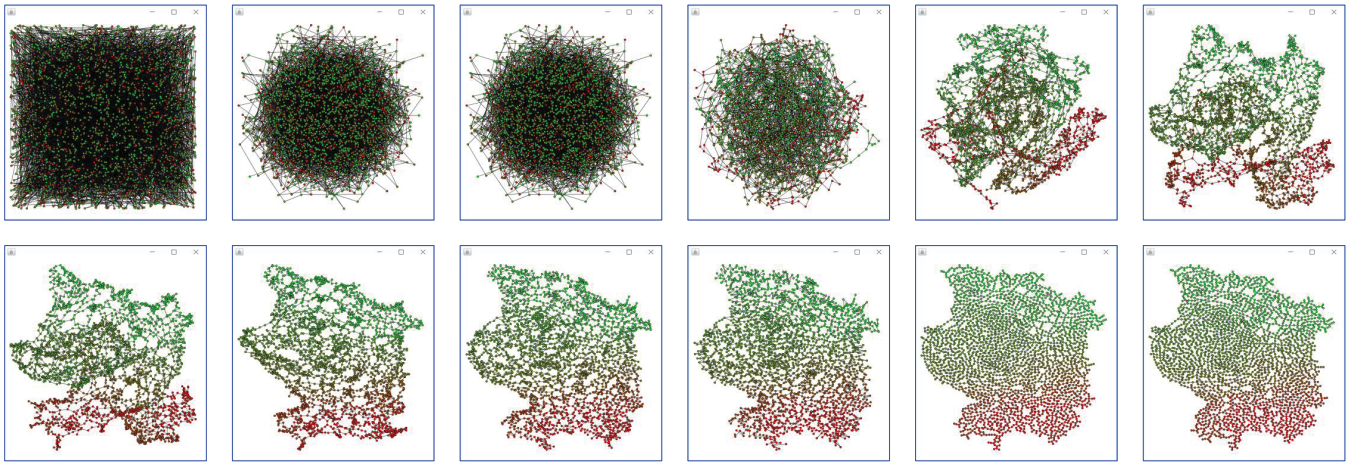

Figure 11: Unfolding of the Minnesota Road Network.

APPENDIX

IMPLEMENTATION AND REPRODUCIBILITY

We provide all code and data to reproduce our experiments at our website [2].

To run the code call `java -jar Gempe.jar` with the following arguments: *source file name, source variable name, number of threads, display(true/false), save intermediate results (true/false).* The meaning of the input parameters is as follows:

(1) *source file name:* name of the Matlab file where the adjacency matrix of the graph is stored
(2) *source variable name:* name of the variable representing the adjacency matrix
(3) *number of threads:* should be set to the number of available cores
(4) *display:* should be set to true or false. If true, windows showing the final result and intermediate results are opened.
(5) *save intermediate results:* If set to true, the results after initialization and all 100 iterations are saved.

Example: `java -jar Gempe.jar meshes.mat eppstein 2 true false`

The Java implementation uses the JUNG graph drawing API for visualization [3]. Please, please use the class `ParallelMain.java` which provides a multi-threaded implementation of GEMPE. This is also the main class which we used to build the `GEMPE.jar`. Set *save intermediate results* to *true* in order to save intermediate results if desired. You can display the intermediate results using the class `MovieDisplayMain.java` later. In this way, you can watch the Minnesota road network to unfold as in Figure xxx. This class supports reading in a numerical class label according to which the nodes should be colored. For the Minnesota data, we used the latitude for each node (which is provided together with the data) to determine the coloring, see the Java code for details. From Figure 11 we can see how GEMPE works. Starting from a uniform random initialization, the road network gradually unfolds and is finally nicely drawn.

The class `DisplayEmbedding.java` displays the current embedding in a window and updates it concurrently. This class is suitable for smaller data sets and you can watch them to unfold, see, e.g., the Eppstein finite element mesh in Figure 12. At the left side of the figure we display the evolution of the objective function which is written to the standard output.

We also provide a highly efficient C++ implementation with OpenMP parallelization and AVX (Advanced Vector Extensions) vectorization, analogously to [1]. It can be found in the folder c++Code. Instructions how to compile the code can be found in the readme File.

All experimental data are open source. Table 2 summarizes their characteristics. The data sets can be obtained at these websites:

- Football https://www.cise.ufl.edu/research/sparse/matrices/Newman/football.html
- Airflights https://sparse.tamu.edu/Pajek/USAir97
- Eppstein and Tapir https://github.com/YingzhouLi/meshpart
- Minnesota http://networkrepository.com/road-minnesota.php
- DBLP https://dblp.uni-trier.de/faq/How+to+parse+dblp+xml
- PubMed https://linqs.soe.ucsc.edu/data
- BlogCatalog http://socialcomputing.asu.edu/datasets/BlogCatalog3
- Cora https://linqs.soe.ucsc.edu/data
- Email EU-Core http://snap.stanford.edu/data/email-Eu-core.html

Graph Drawing: Implementation and Parameter Settings

We compare to the implementations of the graph drawing methods in the Tulip package [4]. We run the algorithms with default parameter settings as implemented in Tulip 5.1.0 as we did not see a systematic improvement of the result when varying the parameters. To summarize the most important parameter settings:

- GEM [7]: 3000 rounds, minimal temperature: 0.005, initial temperature 12, gravitational constant 0.0625, attraction formula Fruchtermann Reingold, and leaving also all other parameters at their default values

[2] https://dm.cs.univie.ac.at/research/downloads/
[3] http://jung.sourceforge.net/
[4] https://tulip.labri.fr/TulipDrupal/

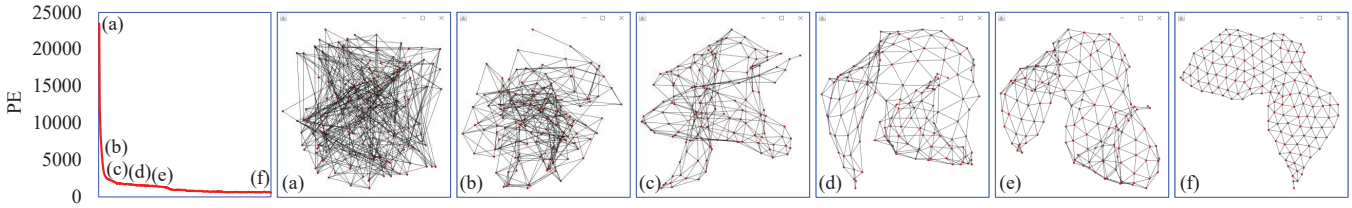

Figure 12: Unfolding of the Eppstein Mesh in the Java Implementation.

- Multipole [13]: 100 iterations, 5 coefficients
- FM3 [15]: threshold 0.01, Galaxy Choice NonUniformProbLowerMass
- GRIP [9]: no input parameters required/supported in Tulip 5.1.0
- LinLog [18]: gravitation factor 0.05, repulsion exponent 0, attraction exponent 1
- SOM [11]: Inverted Self Organizing Maps algorithm implemented in the Jung API http://jung.sourceforge.net/index.html, no input parameters required/supported
- P-MDS [2]: we use 10% of the data as pivot elements
- SPE [21]: For SPE, we used K-nearest neighbors to generate constraints and parameterized the method as recommended by the authors . Using 90% of the generated constraints allowed processing most of our experimental data by the Matlab implementation[5] on a workstation with 8 GB RAM and 2.6 GHz CPU without running out of memory.
- Stress [10]: 200 iterations, edge costs 100
- MulMent [17]: We obtained the c-code by the authors and used standard parameter settings

Graph Representation Learning: Implementation and Parameter Settings

We summarize details about the implementation and parameter settings of the graph representation learning methods. All node classification and node clustering experiments have been performed 100 times and we report the average results. Python scripts for these experiments are included in the dropbox.

- AROPE [28]: python implementation available at https://github.com/elvirast/AROPE.git, parameter settings: degree=3, weights=(0.1, 0.01, 0.001)
- VERSE [24]: implementation at https://github.com/xgfs/verse.git, parameterized with alpha = 0.85, nsamples = 3
- DeepWalk [19]: implementation at https://github.com/phanein/deepwalk.git parameterized with window-size =10, walk length = 40, number-walks = 10
- SDNE [27]: implementation at https://github.com/suanrong/SDNE.git, parameterized with architektur=(-1,1000,128), alpha=100, gamma=1, reg=1, beta=10, batchsize=16, epochs=100, learning rate=0.01, dbnepochs=200, ngsampleratio=0
- node2vec [14]: implementation at https://github.com/xgfs/node2vec-c.git, parameter settings: nwalks=80, walklen=80, window=10, nsamples=5, p=1, q=1, learning rate: default

[5]http://www.cs.columbia.edu/~jebara/code/spe/

	nodes	edges
Football	115	613
Airflights	322	2 126
Eppstein	547	1 566
Tapir	1 024	2 846
Minnesota	2 640	3 302
DBLP	17 228	161 354
PubMed	19 717	44 338
BlogCatalog	10 312	333 983
Cora	2 708	5 429
CiteSeer	3 312	4 732
Email-Eu-Core	1 005	25 571

Table 2: Data Sets.

- struct2vec [20]: implementation at https://github.com/leoribeiro/struc2vec.git, parameter settings: num-walks=10, walk-length=80, window-size=10, all optimizations turned on
- LINE [23]: Implementation available at https://github.com/tangjianpku/LINE, parameter settings: order = 2, samples=1M, rho = 0:025, negative=5
- t-SNE [25]: dimensionality reduction technique commonly used in representation learning literature for visualization. We used it with standard parameter settings: perplexity 20, $\theta = 0.05$ and initial SVD to 30 dimensions followed by 2000 iterations