Master Thesis

Knowledge graph abstraction layer

Author:
Arnaud Guibbert

Supervisors:
Prof. Volkan Cevher
Dr. Daniel Dobos
Vincent Coriou
Andrej Janchevski

Laboratory for Information and Inference Systems
STI Microengineering
Swisscom Digital Lab, Swisscom AG

Lausanne, March 2023

"Committing mistakes is inevitable,
but if one intends to do so,
it is better to make them in even numbers
to ensure they neutralize each other."

1 Acknowledgements

I would like to thank professor Cehver as well as Dr. Dobos for their warm welcome to their labs and to the research teams.

Among my direct supervisor, I would also like to especially thank Vincent who supervised me in Swisscom and helped me go beyond the expected limits and widen my perspective. A special thanks to Nati and Emma for their constant support. A warm thank you for keeping us in good spirits, reminding us to take care of each other and be there for us in moments of stress and doubt.

Thanks as well to the Swisscom FlyingCircus team, especially Milena, Nikola and Samuel, specifically for their support and guidance with the data analysis, as well as their regular feedback. I would also like to thank Andrei Janchevski, who is currently doing his PhD in the lab and was among our direct supervisors. I am grateful for his guidance and support throughout my master's thesis project.

Abstract

This work discusses the use of Knowledge Graphs (KGs) in representing real-world data, which is becoming increasingly popular due to its structured representation of data, including node and edge labels. However, the abundance of information in real-world networks often results in KGs containing a significant number of nodes, making it challenging to obtain insights from the data. The paper focuses on graph-attributed clustering, which aims to partition nodes into groups based on similarity. The work explores the use of deep learning methods, such as Graph Neural Networks (GNN), to improve the generation of embeddings for graph-attributed clustering tasks, and develops a method for generating smaller KGs from an original KG by merging nodes and subgraphs that share similar local structures. The study introduces an unsupervised approach for generating embeddings that satisfy two key requirements: exhibiting a multi-modal distribution and mapping nodes that have similar neighborhood structures to similar embeddings. Secondly, the paper proposes a second module to group the nodes into "Supernodes" based on the clusters provided by the embeddings. Since the scalability of the deep learning methods in a graph context is the bottleneck that limits their usage, a last module is introduced to tackle this issue. Experiments show that the methods reached the objectives initially set and can discover unconventional communities.

Contents

2 Introduction

The use of Knowledge Graphs (KG) as a means of storing real-world data has become increasingly prevalent. This structured representation of data allows for the incorporation of edge and node labels, including relations and features, which enriches the original graph structure. Consequently, KGs are a suitable choice for representing a variety of networks, including social relations, communications networks, and biological entities. However, the abundance of information in real-world networks often results in KGs containing a significant number of nodes, which can prove challenging to handle. Although efficient visualization tools have been developed, as the number of nodes surpasses 10K, obtaining insights becomes increasingly difficult. Therefore, summarizing large KGs with reasonable time and memory complexity while achieving high performance has become a critical task. In this work, we focus on graph-attributed clustering. Graph-attributed clustering aims to partition the set of nodes $\mathcal{V}$ into disjoint groups $\{\mathcal{V}_1, ...\mathcal{V}_G\}$, such that nodes in the same groups are similar with respect to a given distance.

Many fast algorithms are performing graph clustering by finding clique-like communities by maximizing the overall modularity [1, 2], while others try to approximate subgraphs with a set of few meaningful subgraphs [3]. SlashBurn [4] algorithm adopts another approach and revisits the way to define communities. Those algorithms, due to their simplicity, are showing very good performances in terms of time complexity. However, they lack flexibility and cannot well identify all types of communities when those latter are exhibiting a wide diversity.

With the emergence of graph machine learning, the flexibility of the algorithms in discovering unconventional communities have been improved especially with the introduction of the embeddings concept [5, 6]. Spectral clustering [7] generates embeddings that group nodes that are well connected. It exhibited good results in discovering hub-like communities, however, it is not scalable since those methods rely on the diagonalization of the Laplacian matrix. Translational models such as TransE [8] as well as semantic models [9, 10, 11] have shown good results in link prediction and node classification tasks. Though they are initially not designed for graph-attributed clustering, the embeddings they are producing can be used as a basis for other methods. Nevertheless, due to the simplicity of the models, the embeddings generated fail to encapsulate all the information of the nodes and edges.

The generation of embeddings has been hugely improved in the wake of deep learning and especially with the emergence of Graph Neural Networks (GNN). The introduction of Graph Convolutional Networks (GCN) [12, 13] and later of Graph Attention Networks [14] have greatly contributed to improving the performances in graph-related tasks. These methods, in combination with Graph Auto-Encoders (GAE) [15], have enabled the generation of embeddings for a wider range of tasks, especially for graph-attributed clustering. To further improve the performance of auto-encoders for clustering tasks, DAEGC [16] as well as GMM-VGAE [17] aim to impose a cluster-friendly distribution in the auto-encoder latent space, resulting in pre-partitioned nodes. Additionally, ARGA [18] introduces a generative adversarial network on top of the Auto-Encoder to impose any desired distribution in the latent space. Despite the significant advances achieved by deep learning approaches in solving graph-related tasks, they come with a high computational cost and often require the entire graph to be loaded into memory. To mitigate this issue, stochastic methods have been developed. For instance, Cluster-GCN [19] diagonalizes the adjacency matrix into blocks to leverage the message passing trick, while [20] introduces a new sampling method coupled with the normalization technique.

This work aims to develop a method for generating a sequence of increasingly smaller KGs from an original KG, referred to as abstracted KGs. The proposed approach involves merging nodes and subgraphs that share similar local structures, intending to reduce the size of the KGs. To accomplish this, the methodology presented in this work relies on task-tuned embeddings and graph processing. First, an unsupervised approach, inspired by GMM-VGAE [17], is introduced for generating embeddings that are well-suited for graph-attributed clustering tasks. The embeddings are designed to satisfy two key requirements:

Requirement 1. *The latent space should exhibit a multi-modal distribution, i.e. a cluster-friendly distribution. Such an example of a distribution is illustrated on a toy example in Figure 1 and on a real data set in Figure 4.*

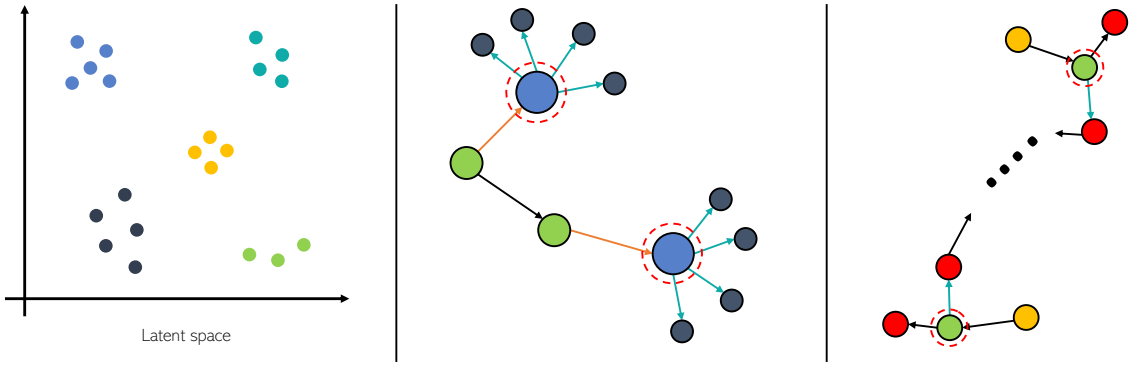

Figure 1: The three requirements we aim at fulfilling with our methods are presented above. Requirement 1 is depicted on the left where the latent space is made of clusters for which intra-cluster distances are much lower than inter-cluster distances. Requirement 2 is illustrated in the middle: the two nodes highlighted with a red circle have the same 1-hop neighborhood and should then be mapped to the same location in the latent space. Requirement 3 states that nodes that are far away should stay relatively far away. Thus on the left sketch, even though green nodes are sharing a similar neighborhood, they should not be merged to preserve the global structure.

Requirement 2. *Two nodes that have exactly the same k-hop neighborhood should be exactly mapped to the same vector/embedding in the latent space. And by extension, two nodes that have a similar neighborhood structure should be close in the latent space. The notion of neighborhood similarity is qualitatively illustrated on Figure 1.*

Based on the clusters provided by the embeddings, we present a second module to group the nodes into **Supernodes**, while preserving the global structure of the graph[1]. The resulting graph, consisting of Supernodes, is referred to as the "abstracted graph". From this we formulate our third and last requirement:

Requirement 3. *The abstracted graph generated from a graph should preserve the global structure of the graph, in terms of graph distances. Qualitatively, if two nodes are far away, respectively close, to each other in the input graph, they must be relatively far away, respectively close, to each other in the abstracted graph.*

An illustration of the three requirements is given in Figure 1. To generate a sequence of abstracted graphs, the output from one iteration is recycled to build the next one. To enable scaling to large graphs, we draw inspiration from the work of Cluster-GCN [19].

This work is organized as follows: section 3 presents the related work as well as the necessary theoretical background for the next sections. The different methods developed in this work and briefly mentioned previously are described in section 4. The description of the data sets, the pre-processing pipeline as well as the metrics of the evaluation stage are presented in section 5. section 6 the different results on the different data sets: it discussed the benefit of the methods and provides metrics to estimate the scaling-up potentialities. The further improvements to be performed, our findings, and conclusions are discussed in section 7 and 8.

[1]The mathematical meaning of *preserving the global structure* will be defined in section 4

3 Related work & Theoretical background

Prior to describing the methods employed in this work, we will provide definitions for the key concepts that are addressed throughout the paper. We will first introduce the relevant notation and terminology in knowledge graph theory, including important tools and techniques. Next, we will review the fundamentals of neural networks, with a focus on Graph Neural Networks (GNNs). Finally, we will introduce a specific type of GNN, the **Variational Graph Auto-Encoder** (VGAE), which will be heavily utilized and expanded upon in section 4.

3.1 Knowledge graph theory

Prior to defining the concept of Knowledge Graphs (KGs), it is necessary to provide a definition of a graph.

Definition 1. *A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a data structure defined by a set of nodes $\mathcal{V} = \{v_1, ..., v_N\}$, with N the number of nodes, and a set of edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$.*

Each node $v_i \in \mathcal{V}$ can be connected to another node $v_j \in \mathcal{V}$, then an edge $e_{i,j} = (v_i, v_j)$ exists between the two nodes, and $e_{i,j} \in \mathcal{E}$. Notice that the edge relation between two nodes is a priori non-symmetric, meaning that $e_{i,j} \in \mathcal{E}$ does not imply that $e_{j,i} \in \mathcal{E}$. If all the edges are symmetric ($\forall i, j \;\; e_{i,j} \in \mathcal{E} \Rightarrow e_{j,i} \in \mathcal{E}$) then the graph is undirected, otherwise the graph is directed. Furthermore, if two edges with the exact same pair of nodes exist, then the graph is a multi-graph (multiple edges), otherwise, it is a simple graph. The graphs processed in this work are multi-directed. A powerful object used to represent a simple graph is the adjacency matrix.

Definition 2. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with N nodes, the adjacency matrix $A \in \{0, 1\}^{N \times N}$ is defined as follows:*

$$A_{i,j} = \begin{cases} 1 & \text{if } \; e_{i,j} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

If the graph is undirected then A is symmetric. A way to represent unweighted multi-graphs with the adjacency matrix is to define $A_{i,j}$ as the number of edges between v_i and v_j. We can now define what is a knowledge graph: qualitatively it is a richer version of a graph.

Definition 3. *A knowledge graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$ is a data structure defined by a set of nodes $\mathcal{V} = \{v_1, ..., v_N\}$, with N the number of nodes, and a set of edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ where $\mathcal{R} = \{r_1, ..., r_R\}$ is the set of relations, with R the number of relations. Each node $v_i \in \mathcal{V}$ has a label $l_i \in \mathcal{L}$.*

The main difference with a simple graph lies in the fact that a knowledge graph has labeled nodes and labeled edges (namely relations). Thus an edge is now defined by a triplet $e_{h,r,t} = (h, r, t)$ (head, relation, tail) where $r \in \mathcal{R}$ and $s, t \in \mathcal{V}$. Notice that it may exist many edges with different relations between two nodes. Similar to what was done with graphs, one can define the relational adjacency matrix object as follows:

Definition 4. *Given a knowledge graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$ with N nodes and R relations, the relational adjacency matrix $\hat{A} \in \{0, 1\}^{N \times R \times N}$ is defined as follows:*

$$A_{h,r,t} = \begin{cases} 1 & \text{if } \; e_{h,r,t} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

The term "relational adjacency matrix" may be misleading, as this object is not a matrix, but rather a tensor of order three. To be able to fully knowledge graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, the relational adjacency matrix has to be combined with the label matrix $\bar{L} \in \{0, 1\}^{N \times L}$, with L the number of labels, defined as follows:

$$\bar{L}_{i,l} = \begin{cases} 1 & \text{if } \; l_i = l \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In section 4, our algorithms will take the KG as input, which will be represented by the tuple $(A, \bar{L})$. Since the KGs processed in this work are multi-graphs, we are redefining the relational adjacency matrix for multi-graphs as:

$$A_{h,r,t} = |e \in \mathcal{E} \;\; \text{s.t.} \;\; e = (h, r, t)|$$

We also define the aggregated relational adjacency matrix $\bar{A} \in \{0,1\}^{N\times N\times R}$ for multi-graphs as follows:

$$\bar{A}_{h,r,t} = \mathbb{1}_{(A_{h,r,t}>0)}$$

The aggregated relational adjacency matrix binarizes the relational adjacency matrix to end up with a simple unweighted KG. The last concept of KG theory needed for the understanding of this work is the notion of k-hop neighborhood.

Definition 5. *Given a knowlegde graph $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{R},\mathcal{L})$, the k-hop neighborhood of a node v is the subgraph $\mathcal{G}(\mathcal{V}_{v,k},\mathcal{E}_{v,k},\mathcal{R},\mathcal{L})$, where:*

$$\mathcal{V}_{v,k} = \{u \mid \text{sp}(v,u) \leq k, u \in \mathcal{V}\}$$

$$\mathcal{E}_{v,k} = \{(h,r,t) \mid h,t \in \mathcal{V}_{v,k}, r \in \mathcal{R}, (h,r,t) \in \mathcal{E}\}$$

Where $\text{sp}(u,v)$ *is the shortest path length between node u and node v*

To put it differently, k-hop neighborhood is to graph what a circle of radius r is to geometry.

3.2 Graph Neural Networks

For the sake of simplicity and also to stick to the material needed to understand this thesis, only the most basic equations (forward pass) of the Multi-Layer-Perceptron (MLP) will be recalled. A wider view of deep neural networks is given in [21].

Definition 6. *Given an MLP with K hidden layers, a hidden dimension P, F input features, and C output features, the forward pass is given by the following sequence of operations:*

$$x^{(0)} \in \mathbb{R}^F, \quad \forall k \in \{1,...,K-1\} \quad x^{(k)}, s^{(k)} \in \mathbb{R}^P, \quad x^{(k)} \in \mathbb{R}^C$$
$$\forall k \in \{1,...,K\} \quad s^{(k)} = W^{(k)}x^{(k-1)} + b^{(k)} \quad x^{(k)} = \sigma(s^{(k)})$$

Where $W^{(k)}$, $b^{(k)}$ are layer-specific trainable weight matrices and bias vectors, and $\sigma(.)$ the activation function (mostly sigmoid *or* ReLU*). Where* ReLU *and* sigmoid *are defined as follows:*

$$\text{ReLU}(x) = \max(x,0) \quad \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

The MLP then takes as input a vector $x^{(0)}$ and outputs a vector $x^{(k)}$ that will be further used to compute a scalar value, commonly called the loss. Then the parameters (namely weight matrices and bias vectors) will be optimized with respect to this loss. The optimization process mostly relies on backpropagation and gradient descent.

A GNN is a specialized class of neural networks that exploits the structure of a graph to perform various tasks such as node prediction, link prediction, and graph clustering. Unlike a traditional neural network, a GNN leverages the graph's topology and takes the graph structure as input to output another graph, probabilities, or other related outputs depending on the specific task. For instance, in the context of node classification with binary labels, a Multilayer Perceptron (MLP) could be used to predict the node labels based on node features X alone, ignoring the relationships between the nodes. However, such an approach ignores the graph structure while a GNN will leverage both A and X to perform more effective node classification.

3.2.1 Graph convolutional network

Graph convolutional network (GCN)[12] is to graph what Convolutional neural network (CNN)[22] is to image. Actually, CNNs inspired GCNs. Graph convolution is an operation using the notion of message passing. For each node in the graph, it qualitatively consists in sending and receiving information to and from its 1-hop neighborhood. The message-passing mechanism may be further illustrated through its equation:

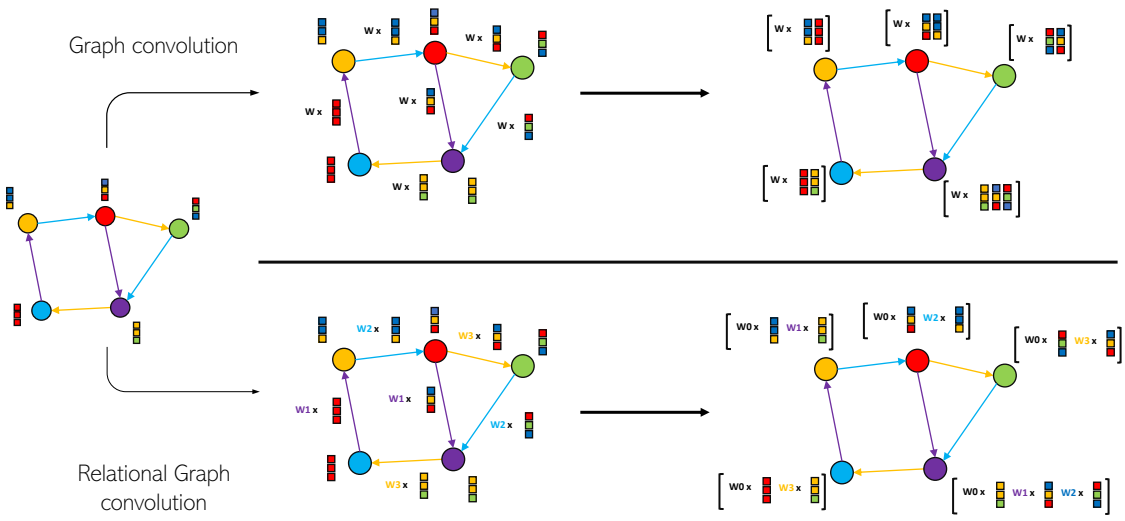

Figure 2: The message-passing mechanism used in both GCN and RGCN modules is illustrated above. The node and edge colors are respectively representing the labels and relations. GCN performs message-passing using the same weight matrix for all the edges, independently of the relations, while RGCN uses a specific weight matrix for each relation, that is to say, three weight matrices plus one for self-loops.

Definition 7. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with N nodes and F input features for each node, the graph convolution operator at layer k performs the following transformation.*

$$h_v^{(k+1)} = \sigma\left(\frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} W^{(k)} h_u^{(k)}\right)$$

where $W^{(k)}$ is a layer-specific trainable weight matrix and $\mathcal{N}(v)$ is the set of neighbors of v including v itself. $\sigma(.)$ denotes the activation function. The full GCN takes as input node features $X \in \mathbb{R}^{N \times F}$ also denoted $H^{(0)}$.

A bias term can be added in addition to the weight matrix multiplication. In addition Figure 2 depicts visually the notion of message passing. As one may notice, the aggregation of the information is performed by taking the mean but other approaches suggest different aggregation such as the concatenation [23].

GCNs have been designed for graphs but can be applied to KGs as well, even though they will not use the relations and labels[2]. An extension of GCNs, called relational graph convolution networks (RGCNs)[13] has thus been developed to fully leverage the relations in the graph.

3.3 Relational Graph convolutional network

The relational graph convolution operator is very similar to the graph convolution operator. The only difference is that instead of having one weight matrix $W^{(k)}$ per layer, the RGCN uses as many weight matrices $W_r^{(k)}$ as there are relations in the KG. Thus when the information is passed from one node to another, it is processed with a weight matrix that is specific to the relation linking the two nodes. The mathematical definition for the relational graph convolution operator is given by:

[2]the labels can be leveraged if the label matrix $\bar{L}$ is used as input feature X in the GCN.

Definition 8. *Given a knowledge graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$ with N nodes and F input features for each node, the relational graph convolution operator at layer k performs the following transformation.*

$$h_v^{(k+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{u\in\mathcal{N}_r(v)}\frac{1}{|\mathcal{N}_r(v)|}W_r^{(k)}h_u^{(k)} + W_0^{(k)}h_v^{(k)}\right)$$

Where $W_r^{(k)}$ is the layer-specific trainable weight matrix for relation r and layer k, $\mathcal{N}_r(v)$ is the set of neighbors of node v that are connected to it with relation r. The matrix $W_0^{(k)}$ is a layer-specific matrix for self-loop edges. $\sigma(.)$ denotes the activation function.

Figure 2 visually illustrates the difference between RGCN and GCN. Though the number of weight matrices is multiplied by R, in the RGCN configuration, it increases the expressive power of the GNN and fully leverages the information provided by the graph structure.

The two operators presented previously are among the most basic ones used in GNNs, they are the building blocks of a GNN and are often combined with other modules such as MLP, ResNets... With the emergence of transformers in the NLP field, similar operators have been developed in the graph field as well such as the Graph Attention Networks (GAT) [14] and its extension to KG, namely the Relational Graph Attention Networks (RGAT) [24]. Even though they will not be used in this work, essentially because the methods are already working well with basic RGCNs, using such enhanced modules could significantly improve the performances presented in section 6. This point will be tackled in section 7.

3.4 Variational Graph Auto-Encoder

To solve the main task described previously in section 2, the Variational Graph Auto-Encoder (VGAE) [15] architecture has been selected. The ability of this latter to generate embeddings under constraints in an unsupervised way, pushed us to select it. Prior to explaining the core notions of the VGAE, it is necessary to first define what is a Graph Auto-Encoder (GAE). The Graph Auto-Encoder (GAE) will be defined specifically for graphs, but it should be noted that all the definitions presented will also apply to KGs.

Definition 9. *A GAE is a function that takes as input a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with N nodes and returns another graph $\hat{\mathcal{G}}(\hat{\mathcal{V}}, \hat{\mathcal{E}})$, namely the reconstructed graph. The GAE is divided into two parts: an encoder function f and a decoder one g such that:*

$$\begin{aligned} \textit{Encoder} \quad & f : \mathcal{G} \to \mathcal{Z} \\ \textit{Decoder} \quad & g : \mathcal{Z} \to \mathcal{G} \\ \textit{Auto-Encoder} \quad & \hat{\mathcal{G}}(\hat{\mathcal{V}}, \hat{\mathcal{E}}) = g \circ f(\mathcal{G}(\mathcal{V}, \mathcal{E})) \end{aligned}$$

Where $\mathcal{Z}$ is called the latent space and is often $\mathbb{R}^{N\times D}$ with D the dimension of the latent space. A vector is associated with each node, namely an embedding. The common objective of an Auto-Encoder is to approximate the identity function while having D as small as possible.

In the context of Graph Deep Learning, f and g are replaced by GNNs[3]. As a graph's properties can be entirely captured by its adjacency matrix, the encoder function f will accept the adjacency matrix as input, while the decoder function g will produce another adjacency matrix, namely the reconstructed adjacency matrix. In addition to A, the encoder also commonly takes as input a node feature matrix $X \in \mathbb{R}^{N\times F}$. This matrix will be the starting point for like-GCNs modules, i.e. $H^{(0)}$.

A possible simple architecture for the encoder could be a GCN followed by an MLP as depicted in Figure 3. The MLP will output for each node v_i a vector $z_i \in \mathbb{R}^D$, we say that the node has been projected into the latent space. This vector is commonly called the embedding of the node. The embeddings of all the nodes are then concatenated into the embeddings matrix $Z \in \mathbb{R}^{N\times D}$. The decoder g is reconstructing the adjacency matrix and the node features using matrix Z. The GAE presented in Figure 3 is using a regressor[4] to reconstruct the node features and an inner product between the embeddings to reconstruct the adjacency matrix in a probabilistic way. This means that the model will output for each pair of nodes a probability indicating the likelihood of an edge existing between the two nodes. The equations of the full GAE presented in Figure 3 are summed up below:

[3] g will actually be closer to a simple neural network since it will not use any modules that are specific to graphs

[4] note that if the node features are categorical features, it will then use a classifier.

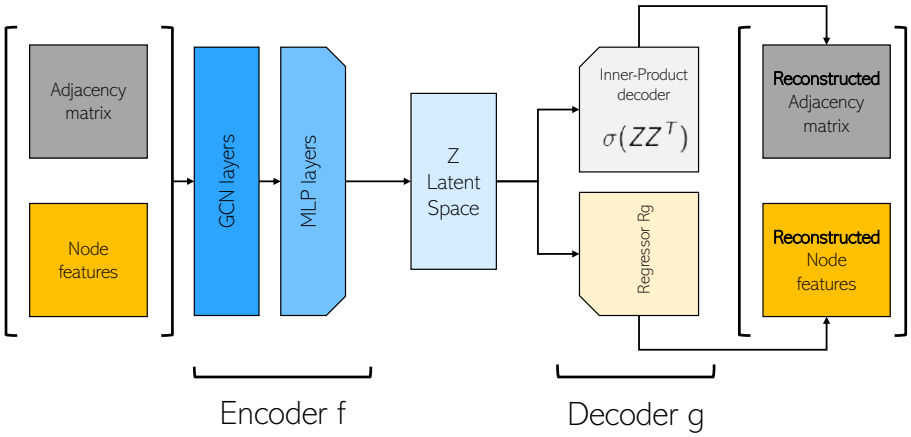

Figure 3: The architecture of a common GAE is presented above. It takes as input the adjacency matrix A and the node features X. The encoder f sequentially applies graph convolution and MLP-like operations to output an embedding for each node. Those embeddings are then used by the decoder g to reconstruct the adjacency matrix and the node features.

$$f = \text{MLP}_f(\text{GCN}(.)) \quad f(A, X) = Z = \begin{pmatrix} {z_1}^T \\ {z_2}^T \\ \vdots \\ {z_N}^T \end{pmatrix} \tag{2}$$

$$\hat{A}, \hat{X} = g(Z) \quad \hat{A}_{i,j} = \sigma(z_i^T z_j) \quad \hat{X} = \text{MLP}_g(Z)$$

Where $\sigma(.)$ is the sigmoid function. Even though Auto-Encoders were developed before the advent of deep learning, they have been boosted by this latter due to the expressive power provided by deep neural networks. The Variational Auto-Encoder (VAE) [25] is an extension of the Auto-Encoders[5]. As usual, it has its graph form, namely the VGAE [15].

One way to explain the goal of the VGAE is the following: in the classical GAE, the latent space does not have any explicit constraints and can actually have any kind of distribution as soon as it reconstructs well the graph. The VGAE makes an assumption on the distribution of the vectors in the latent space. More specifically, it assumes that the graphs have been generated following a **generative process** where the starting point is a vector in the latent space. It assumes that starting from this vector we can reconstruct a graph that is very likely to be in our data set of graphs (high likelihood). To go from a vector to a graph, the decoder function g, which is actually part of the generative process, is used. The classical VGAE is assuming a normal distribution $\mathcal{N}(0, I)$ for the latent space structure. But how is it possible, without any more assumptions on the graph data set, to have such a strong assumption on the latent space? Well, this is because any continuous distribution in D dimensions can be generated by taking a set of D variables that are normally distributed and mapping them through a sufficiently complicated function. The sufficiently complicated function will be g, and since it is a deep neural network, its complexity can be easily increased. The trainable part of g will actually be trained such that we are matching the distribution of our graph data set, in other words, such that we are maximizing the likelihood. To be able to better define the likelihood in the VGAE case, it is necessary to describe the generative process of the classical VGAE [15].

[5] A VAE is actually a generative model and is mostly related to the Auto-Encoder because of its encoder/decoder-like structure

Definition 10. *The generative process of the VGAE with an inner product decoder for the adjacency matrix and a regressor r_g for the node features is the following one:*

- $\forall i \in \{1, ..., N\}$ *choose a vector* $z_i \sim \mathcal{N}(0, I_D)$
- $\forall (i, j) \in \{1, ..., N\} \times \{1, ..., N\}$ *compute the expectation for the existence of the edge*

$$p_{i,j} = p(A_{i,j} = 1 | z_i, z_j) = \sigma({z_i}^T z_j)$$

And sample from $A_{i,j} \sim \mathcal{B}(p_{i,j})$ *to determine if the edge exists.*

- $\forall i \in \{1, ..., N\}$ *Compute the expectation for the value of the node feature*

$$\mu_{i,x}, \log(\sigma_{i,x}) = r_g(z_i)$$

And sample from $x_i \sim \mathcal{N}(\mu_{i,x}, I\sigma_{i,x}^2)$ *to determine the value of the node feature.*

This way, one can generate a new graph $\mathcal{G}'$. To be able to maximize the likelihood of this generative process with respect to our graph data set, a probability model is first needed, that is to say, a tool to compute $p(A, X)$. Unfortunately $p(A, X)$ cannot be directly accessed but using the law of total probability, it can be theoretically computed:

$$p(A, X) = \int_z p(A, X|Z)p(Z)dZ$$

Although the terms in the integral are known, it is unfeasible to compute the integral in an analytical form due to the complexity of the decoder function g. Thus, the only practical solution is to approximate the integral. One possible way to do this is by using an unbiased estimator such as the following one:

$$\hat{p}(A, X) = \frac{1}{n} \sum_{i=0}^{n} p(A, X|Z_i)$$

With $\{Z_1, ..., Z_n\}$ sampled in the latent space. As suggested in [26], it will require an untractable number of samples before our estimator will be closed to the true value of $p(A, X)$ because the $Z_i s$ selected are not likely at all to produce likely graphs. To circumvent this, an inference model is introduced to generate Zs that are producing highly likely graphs. This latter is denoted $q(Z|A, X)$ and we are then trying to maximize the expectation of $p(A, X|Z)$ under $q(Z|A, X)$. Beyond the intuition, the mathematical reason to do so comes from the fact that $p(A, X)$ can be lower bounded by $\mathbf{E}_{Z \sim q(Z|A,X)}[\log(p(A, X|Z))]$ such that:

$$\log[p(A, X)] \geq \mathbf{E}_{Z \sim q(Z|A,X)}[\log(p(A, X|Z))] - D_{KL}[q(Z|A, X)||p(Z)]$$

Where D_{KL} is the Kullback-Leibler divergence. This is the corner point of the VGAE, and this lower bound is known as the **evidence lower-bound** (ELBO). This is actually an elegant way to maximize $p(A, X)$ without having to explicitly compute it. The last point to define the way q is built. Going back to the GAE, q will actually be a top layer on the encoder function f. Indeed in the GAE, f outputs a single vector Z in the latent space, now it will output a mean and a standard deviation having the same dimension of Z that will be used to parametrize $q(.|A, X)$ such that:

$$q(Z|A, X) = \prod_{i=1}^{N} q(z_i|A, X) \quad \text{where} \quad q(z_i|A, X) = \mathcal{N}\big(z_i; \mu_i, \text{diag}(\sigma_i^2)\big)$$

$$\mu, \log(\sigma) = f(A, X)$$

Where μ_i and $\log(\sigma_i)$ are concatenated in μ and $\log(\sigma)$, in the same way it is done with z_i and Z in Equation 2. Finally putting all equations together, the VGAE optimization problem is given by:

$$\begin{aligned} &\min_{f,g} \quad \mathbf{E}_{X,A \sim p(A,X)}[\mathcal{L}_{ELBO}(X, A)] \\ &\text{with} \quad \mathcal{L}_{ELBO}(A, X) = D_{KL}[q(Z|A, X)||p(Z)] - \mathbf{E}_{Z \sim q(Z|A,X)}[\log(p(A, X|Z))] \end{aligned} \tag{3}$$

The last important point to define for the VGAE is the way it is optimized. The optimization of the encoder and decoder parameters is done via stochastic gradient descent, which requires the loss to be differentiable with respect to their trainable parameters. As one can notice, the ELBO loss $\mathcal{L}_{ELBO}$

is made of two parts: the KL divergence loss, which is straightforward to compute for two Gaussian distributions, and the second term $\mathbf{E}_{Z\sim q(Z|A,X)}[\log(p(A,X|Z))]$. This one is way more tricky since we first need to sample Z using $q(.|A,X)$ and then apply the decoder to it. The sampling operation unfortunately breaks the computational graph and makes the loss non-differentiable with respect to the parameters of the encoder f. To solve this issue, one can use the **reparametrization trick** which consists in replacing $z \sim \mathcal{N}(\mu, I\sigma^2)$ by $z = \sigma\epsilon + \mu$ with $\epsilon \sim \mathcal{N}(0, I)$. This way z is still sampled from $\mathcal{N}(\mu, I\sigma^2)$ while not breaking the computational graph.

The most basic principles of the VGAE have been defined and will be used to define variations of the classic VGAE in section 4. To get further insights and exhaustive explanations/proofs, please refer to [26, 25, 15].

3.5 Gaussian Mixture Models

The last model, widely used in this work to be able to generate cluster-friendly embeddings, is Gaussian Mixture Model (GMM). GMM is a soft clustering algorithm, where each point "belongs" to all clusters with a given probability. Given a set of data points $\{x_1, ..., x_M\}$ where $x_i \in \mathbb{R}^F$, GMM aims to fit the distribution of the data points with a mixture of Gaussians. More precisely it assumes that the data points have been generated following the generative process described below:

Definition 11. *A Gaussian Mixture Model with C Gaussians assumes that each data point x_i has been generated the following way:*

- *For each data point i, select a Gaussian c_i such that $c_i \sim \text{Cat}(\pi)$, with $\pi = (\pi_1, ..., \pi_C)$ namely the mixture weights*
- *Generate the data point x_i using the Gaussian selected such that $x_i \sim \mathcal{N}(\mu_{c_i}, \Sigma_{c_i})$*

Where π_c represents the probability of having been generated by gaussian c, and μ_c, Σ_c being respectively the mean and covariance matrix of Gaussian c.

The objective of a GMM is then to find parameters $\theta = \{\pi_1, ..., \pi_C, \mu_1, ..., \mu_C, \Sigma_1, ..., \Sigma_C\}$ such that the likelihood on the data set is maximized. The log-likelihood under the GMM can be computed as follows:

$$\begin{aligned}
\log(\mathcal{L}(x_1, ..., x_M)) &= \sum_{i=1}^{M} \log\Big(p(X_i = x_i|\theta)\Big) \\
&= \sum_{i=1}^{M} \log\Big(\sum_{c=1}^{C} p(X_i = x_i|c_i = c, \theta)p(c_i = c|\theta)\Big) \\
&= \sum_{i=1}^{M} \log\Big(\sum_{c=1}^{C} \mathcal{N}(x_i; \mu_c, \Sigma_c)\pi_c\Big)
\end{aligned} \tag{4}$$

One is finally ending up with a sum inside a logarithm and, unfortunately finding a general closed-form solution for the maximum of this log-likelihood is not possible. To circumvent this issue the GMMs uses the **Expectation Maximization** (EM) principle.

3.5.1 Expectation Maximization

Let's assume one wants to maximize a complex function $g(\theta)$ over θ for which you can derive a simpler function as a lower bound $\hat{g}(\theta, \theta_t)$ that satisfies the following assumptions:

$$\forall\theta \quad g(\theta) \geq \hat{g}(\theta, \theta_t) \quad \text{and} \quad g(\theta_t) = \hat{g}(\theta_t, \theta_t) \tag{5}$$

Where θ_t is a fixed set of parameters. Let's also assume that it is possible to derive a closed-form solution for the maximum of $\hat{g}(\theta, \theta_t)$. Then given a state θ_t the EM iterative procedure computes θ_{t+1} as follows.

- **Expectation step**: compute the terms related to θ_t function $\hat{g}(\theta, \theta_t)$.
- **Maximization step**: compute θ for which $\hat{g}(\theta, \theta_t)$ is maximal and update θ_{t+1}. That is to say:

$$\theta_{t+1} = \arg\max_{\theta} \hat{g}(\theta, \theta_t)$$

  Using Equation 5, the following inequality holds : $g(\theta_{t+1}) \geq g(\theta_t)$.

Algorithm 1 Expectation maximization GMM

Input:
Initialize parameters θ_0, Maximum number of iterations T, $\{x_1, ..., x_N\}$
Initialization:
$t = 0$
while $t < T$ **do**
Expectation step: compute $\gamma_{i,c}^{(t)}$
Maximization step: compute θ_{t+1} using Equation 6
if $\theta_t = \theta_{t+1}$ **then**
$\theta_T = \theta_{t+1}$
break
end if
$t = t + 1$
end while
Return:
θ_T

Notice that the EM procedure does not guarantee finding the global maximum. To apply EM to the log-likelihood function introduced in Equation 4, one first needs to derive a lower bound satisfying assumptions of Equation 5. Given a set of parameters θ_t, this can be done using Jensen's inequality:

$$\sum_{i=1}^{M} \log\Big(\sum_{c=1}^{C} p(X_i = x_i | c_i = c, \theta) p(c_i = c|\theta)\Big) \geq \sum_{i=1}^{M}\sum_{c=1}^{C} \gamma_{i,c}^{(t)} \log\Big(\frac{p(X_i = x_i, c_i = c|\theta)}{\gamma_{i,c}^{(t)}}\Big)$$
$$\text{where} \quad \gamma_{i,c}^{(t)} = p(X_i = x_i, c_i = c|\theta_t)$$

One can verify that this lower bound satisfies Equation 5, and thus can be used for the EM process. The closed-form solution of the maximum θ_{max} is given below:

$$\mu_c^{(t+1)} = \frac{\sum_{i=1}^{M} x_i \gamma_{i,c}^{(t)}}{\sum_{i=1}^{M} \gamma_{i,c}^{(t)}} \quad \Sigma_c^{(t+1)} = \frac{\sum_{i=1}^{M} \gamma_{i,c}^{(t)} \Big(x_i - \mu_c^{(t+1)}\Big)\Big(x_i - \mu_c^{(t+1)}\Big)^T}{\sum_{i=1}^{M} \gamma_{i,c}^{(t)}} \quad \pi_c^{(t+1)} = \frac{\sum_{i=1}^{M} \gamma_{i,c}^{(t)}}{M} \tag{6}$$

The pseudo-algorithm used to find the best parameters θ for a Gaussian mixture model is described in algorithm 1. Once the best parameters have been found, one still needs to assign to each data point a cluster label, namely determining which Gaussian was the more likely to have generated a given data point. To do so, one chooses Gaussian c for which $p(c|x_i)$, which can be computed as follows:

$$c_i = \arg\max_c p(c|x_i) = \arg\max_c \frac{p(x_i|c)p(c)}{p(x_i)} = \arg\max_c \frac{\pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c)}{\sum_{j=1}^{C} \pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j)} \tag{7}$$

4 Methods

Since we mostly deal with knowledge graphs in this work and for the sake of simplicity, from this point on and for the rest of the work, unless otherwise specified, graph will mean knowledge graph. Recall from section 2 that this work aims at building a sequence of graph increasingly small, called abstracted graphs. To do so, task-tuned embeddings are generated in an unsupervised way to partition the nodes into different clusters. The generation process of the embeddings is presented in subsection 4.1. The way the node clustering is leveraged to build a first abstracted graph is tackled in subsection 4.2. Finally, the iterative process to build a sequence of abstracted graphs in a straightforward manner is described in subsection 4.3. In addition, we present a module, in its beta version, to be applied on the top of the modules to make them scalable to large graphs.

4.1 Variational Graph Auto-Encoder with Gaussian mixture

Recall that the embeddings we intend to generate must fulfill the following two requirements: first, the latent space should exhibit a cluster-friendly distribution, and second two nodes with similar neighborhoods should be mapped to the same embeddings. We will satisfy the first requirement using a VGAE with a specific generative process that assumes that the node's embeddings are coming from a multi-modal distribution. The second one will be satisfied by selecting a specific architecture for the encoder of the VGAE.

4.1.1 Latent space structure

The objective is to obtain a latent space that exhibits a multi-modal distribution. To achieve this goal, the VGAE structure can be used given a multi-modal distribution. In modeling the latter, a GMM is preferred over fixed or unparameterized distributions due to its parametrized nature, which allows for ease in optimization using the EM algorithm described in algorithm section 3. The generative process of the VGAE used to generate the embeddings is the following one:

Definition 12. *The generative process of the VGAE with a MLP decoder for the adjacency matrix, a classifier C_g for the node features and a GMM with C classes and diagonal covariance matrices to model the latent space distribution is the following one:*

- *For each node $i \in \{1, ..., N\}$, select a Gaussian c_i such that $c_i \sim \text{Cat}(\pi)$*
- *For each node $i \in \{1, ..., N\}$ choose a vector in the latent space $z_i \sim \mathcal{N}(\mu_{c_i}, \text{diag}(\sigma^2_{c_i}))$*
- *$\forall (h, r, t) \in \{1, ..., N\} \times \{1, ..., R\} \times \{1, ..., N\}$ compute the expectation for the existence of the edge (h, r, t)*

  $$p_{h,r,t} = p(A_{h,r,t} = 1 | z_i, z_j) = \sigma(\text{MLP}_g(z_h || z_t)_r) \tag{8}$$

  And sample from $A_{h,r,t} \sim \mathcal{B}(p_{h,r,t})$ to determine if the edge exists.
- *$\forall i \in \{1, ..., N\}$ Compute the expectation for the value of the node feature*

  $$p_{i,l} = p(x_i = l | z_i) = C_g(z_i)_l \tag{9}$$

  And sample from $x_i \sim \text{Cat}(C_g(z_i))$ to determine the categorical value of the node feature.

Where the MLP decoder takes as input the concatenation of the embeddings of the head and the tail of an edge and outputs for each relation a probability of existing.

This generative process assumes that each node belongs to a cluster and the objective is to find those clusters. Instead of using an inner-product decoder, this VGAE uses an MLP decoder to increase the expressive power of the decoder and to break the symmetry (i.e. $p_{h,r,t} \neq p_{t,r,h}$) since the graphs are directed. To build the node features, a classifier is used since those ones are categorical in this work. Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, we will actually use the label matrix defined in Equation 1 for the node features (i.e. $X = \bar{L}$).

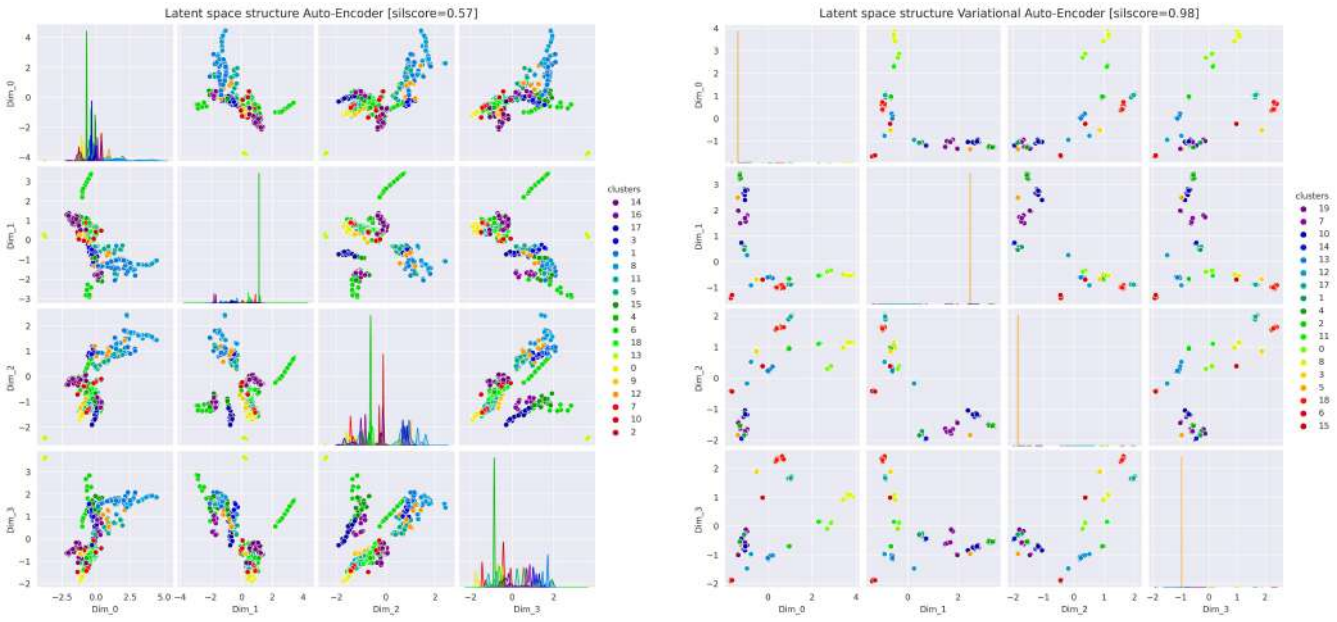

Figure 4: The left, respectively right, pairplot depicts the structure of the latent space (4/32 first dimensions) of the GAE, respectively VGAE combined with a GMM, after 300 epochs of training. The experiments were performed on SCM2K data set. The data points are colored according to their respective cluster labels. Regarding the GAE, a GMM is applied after the training stage to determine cluster labels. The right plot shows clear small globular clusters while the left one appears to be slightly disorganized. This observation is supported by the silhouette score, which is higher for the right plot (0.98) compared to the left (0.55).

The inference model of the VGAE should output cluster labels c_i and associated vectors z_i that will generate very likely graphs. It is denoted $q(Z, c|A, X)$, where c is the concatenation of the c_i predicted for each point, and it is assumed that:

$$q(Z, c|A, X) = q(Z|A, X)q(c|A, X) \tag{10}$$

It means that $Z|A, X$ is independent of $c|A, X$, this assumption will be further discussed n section 7. The inference model is parameterized by the output of the encoder f such that:

$$q(Z|A, X) = \prod_{i=1}^{N} q(z_i|A, X) \quad \text{where} \quad q(z_i|A, X) = \mathcal{N}\big(z_i; \mu_i, \text{diag}(\sigma_i^2)\big)$$

$$q(c|A, X) = \prod_{i=1}^{N} q(c_i|A, X) \quad \text{where} \quad q(c_i|A, X) = \text{Cat}(\pi_i^*)$$

$$\mu, \log(\sigma), \Pi^* = f(A, X)$$

Where μ, σ, Π^* are respectively the concatenation of the vectors μ_i, σ_i, π_i^* predicted for each node.[6] Given this VGAE, the first question one should ask is: does such a VGAE produces a different latent space from what one would get with a GAE for instance? Though this question will be answered in section 6, we are providing first visual results inFigure 4 to show the difference in terms latent space structure. Prior to handling the question of the parallel training of the VGAE and the GMM, let's first describe the architecture used for the VGAE.

4.1.2 Architecture

The whole architecture of the VGAE is illustrated in Figure 5 and as stated before, the node features X is replaced by the label matrix $\bar{L}$. This architecture, especially the encoder one, has been specifically designed such that the embeddings are fulfilling requirement 2, namely that two nodes with similar neighborhoods should be mapped to the same embeddings. Let's give a definition to the notion of node neighborhood.

[6] Actually Π^* will be rather infered from μ_i than output by f.

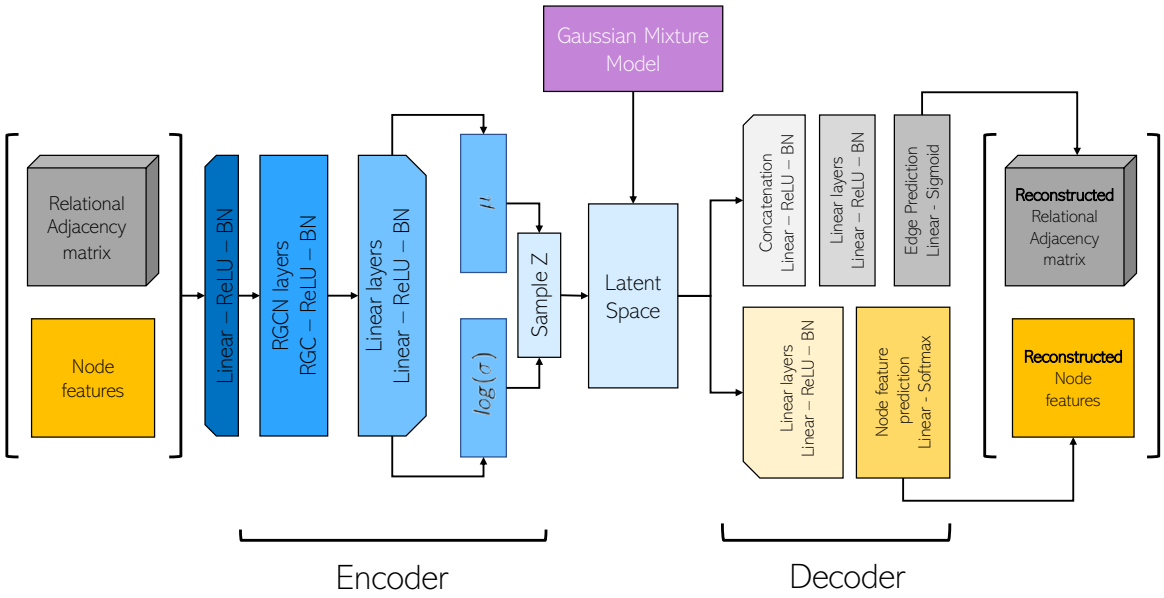

Figure 5: The architecture of the VGAE combined with a GMM is illustrated above. The encoder is made of a linear layer to convert one hot vectors into continuous ones, then a RGCN block followed by a MLP block is applied. Using the reparametrization trick, random vectors z_i are generated to feed both the GMM and the decoder. The decoder has two parts, a first part that reconstructs the aggregated relational adjacency matrix and a second that reconstructs the node features.

Definition 13. *Given a node v in a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, its in-neighborhood of order k is recursively defined as:*

$$\mathcal{N}_{in,v}^{k} = \bigcup_{(h,r,v)\in\mathcal{E}} \{(k, r, \mathcal{N}_{in,h}^{k-1}, h_h), 1)\}$$

Where $\mathcal{N}_{in,u}^{0} = h_u$ with h_u being the node feature associated with node u. Its out-neighborhood is symmetrically defined as:

$$\mathcal{N}_{out,v}^{k} = \bigcup_{(v,r,t)\in\mathcal{E}} \{(k, r, \mathcal{N}_{out,t}^{k-1}, h_t), -1)\}$$

Where $\mathcal{N}_{out,u}^{0} = h_u$. And its neighborhood is defined as:

$$\mathcal{N}_{v}^{k} = \mathcal{N}_{out,v}^{k} \cup \mathcal{N}_{in,v}^{k}$$

$\mathcal{N}_{in,v}^{k}$ only counts in-edges while $\mathcal{N}_{out,v}^{k}$ only counts out-edges. The interesting property of the RGCN module is the following:

Theorem 1. *Given a RGCN with k layers, and two nodes u, v from a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, the following property holds:*

$$\mathcal{N}_{in,v}^{k} = \mathcal{N}_{in,u}^{k} \quad \text{and} \quad h_u = h_v \quad \Rightarrow \quad \text{RGCN}(h_v) = \text{RGCN}(h_u)$$

Where $h_u, h_v \in \mathbb{R}^F$ are respectively the feature vectors of nodes v and u.

This property is almost the one we are looking for our embeddings. Unfortunately it only counts the in-neighborhood and not the out-one. To do so, we are using a trick in the pre-processing of the graph, namely the relations are doubled. That is to say for each relation r, an inverse relation r' is created and each edge (h, r, t) is doubled with an edge (t, r', h). Then the following property holds:

Theorem 2. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$ for which the relations and edges have been doubled, the following equivalence holds:*

$$\forall u, v \in \mathcal{V} \quad \mathcal{N}_{in,v}^{k} = \mathcal{N}_{in,u}^{k} \;\Leftrightarrow\; \mathcal{N}_{v}^{k} = \mathcal{N}_{u}^{k}$$

Combining Theorem 1 and Theorem 2, the embeddings generated by a RGCN with k layer finally fulfill requirement 2. The choice of the encoder architecture depicted in Figure 5 was actually motivated by this property of the RGCN. One may wonder why the RGCN block is surrounded by two MLP-like

modules. The presence of the downstream MLP is related to the form of the input data: the node features are one-hot encoded vectors. To be able to fully leverage the matrices of each relation in the RGCN, the one-hot encoded vectors are transformed into continuous vectors through the downstream MLP. The objective of the upstream MLP is to increase the expressive power of the encoder. This MLP will output μ, Σ and Π^* will be derived using μ, Σ. Indeed getting inspiration from similar works on VAEs combined with GMM [27], for each node π_i^* will be a one-hot encoded vector computed as follows:

$$\pi_{i,c}^* = \begin{cases} 1 & \text{if } \ c = \arg\max_{u \in \{1,\dots,C\}} p(u|\mu_i) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Another model for Π^* is presented in section 7. The reparametrization trick is used to compute the latent vectors z_i, with which the decoder will be fed and output the reconstructed aggregated adjacency matrix and the reconstructed node features.

4.1.3 Training procedure

Let's dive into the core part of this VGAE: the training procedure. The difficulty lies in the fact that we are trying to optimize two modules, namely the Auto-Encoder and the GMM, simultaneously while using two different optimization methods, namely gradient descent and expectation maximization. To do so, an alternating step-like algorithm will be used. We will alternate between gradient descent and expectation maximization.

VGAE training

Regarding the gradient descent one first needs to compute the ELBO loss defined in Equation 3. The KL term can be computed as follows:

$$\begin{aligned} \mathcal{L}_{KL}(A, X) &= D_{KL}[q(Z, c|A, X)||p(Z, c)] \\ &= \sum_{i=1}^{N} \left[D_{KL}[\pi_i^*||\pi] + \sum_{c=1}^{C} \pi_{i,c}^* D_{KL}[\mathcal{N}(\mu_i, \text{diag}(\sigma_i^2))||\mathcal{N}(\mu_c, \text{diag}(\sigma_c^2))] \right] \end{aligned} \tag{12}$$

The derivation of this result is given in the appendix. The second term in the ELBO loss can be approximated by performing S repetitions such that:

$$\begin{aligned} \mathbf{E}_{Z \sim q(Z|A,X)}[\log(p(A, X|Z))] &\approx \frac{1}{S} \sum_{s=1}^{S} \log(p(A, X|Z = Z_s)) \\ &= \underbrace{\frac{1}{S} \sum_{s=1}^{S} \log(p(A|Z = Z_s))}_{\mathcal{L}_{relations}(A)} + \underbrace{\frac{1}{S} \sum_{s=1}^{S} \log(p(X|Z = Z_s))}_{\mathcal{L}_{labels}(X)} \end{aligned}$$

Using the independence of the random variables A and X, the loss can be split into two terms such that the first one corresponds to the reconstruction of the relational adjacency matrix while the second one is related to the reconstruction of the node features, namely label prediction. Since it would be untractable to compute the probability $p_{h,r,t}$ defined in Equation 8, for all edges $(h, r, t) \in \mathcal{E}$, we will use positive and negative sampling. For each gradient step, a set of positive and negative edges, $\mathcal{E}_{pos}$ and $\mathcal{E}_{neg}$, will be sampled such that $\log(p(A|Z))$ will be approximated as follows:

$$\log(p(A|Z)) = \sum_{(h,r,t) \in \mathcal{E}_{pos}} \log(p_{h,r,t}) + \sum_{(h,r,t) \in \mathcal{E}_{neg}} \log(1 - p_{h,r,t})$$

Regarding the second term, using $p_{i,l}$ notation defined in Equation 9 $\log(p(X|Z))$ can be expressed as follows:

$$\log(p(X|Z)) = \sum_{i=1}^{N} \log(p_{i,l_i})$$

Following the intuition and the work on the β-VAE [28], the loss to be optimized will be a weighted linear combination of the losses presented previously. Given A and X the loss is equal to:

$$\mathcal{L}_{VGAE}(A, X) = w_{KL}\mathcal{L}_{KL}(A, X) + w_{relations}\mathcal{L}_{relations}(A) + w_{labels}\mathcal{L}_{labels}(X) \tag{13}$$

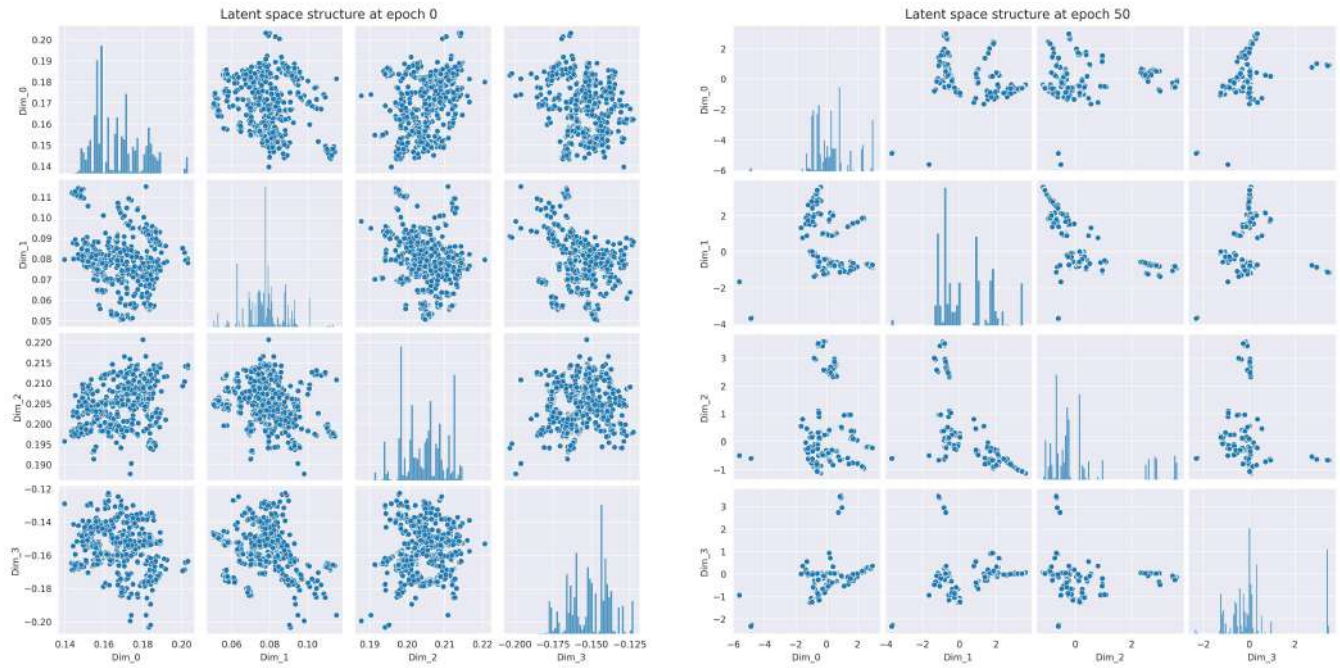

Figure 6: The left pairplot, respectively right, depicts the latent space structure (4/32 dimensions) at the initialization, respectively after 50 epochs, of the VGAE. It exhibits a difference in terms of distribution and scale. At the beginning of the process, the embeddings are confined within a ball of radius less than 0.3, but after 50 epochs, they are dispersed more widely throughout the space. Thus initializing the clusters, before any pre-training operations, would strongly impact the reconstruction loss.

Alternating steps

The GMM will be incorporated into the training of the VGAE using the alternating steps principle. In short, one will freeze GMM parameters to perform x gradient steps, and then freeze the Auto-Encoder parameters to perform y EM steps. In VaDE [27], they do not even update the GMM along the training. Two questions arise: how the periods x and y should be chosen? How are set the initial cluster centers in the GMM? The first point will be mostly discussed in section 5 but, it has been empirically noticed that x should be greater than y since modifying the cluster centers have a bigger impact on the KL loss defined in Equation 12, than the modifications of the embeddings on the log-likelihood.

Regarding the initialization of the cluster centers, the same method used in VaDE [27] was selected, namely that the Auto-Encoder will be first pre-trained, then the data points will be projected into the latent space, such that the GMM will be initialized. The reason behind this selection was primarily influenced by the concern that during the initial stages of training, the reconstruction term might be too weak, leading to the model getting stuck in an undesirable local minimum or saddle point, both of which are difficult to overcome. This is actually confirmed by the observations as shown in Figure 6. It highlights the fact that the latent space evolves very rapidly at the beginning of the training which justifies the delay of the GMM initialization.

During a few epochs, the latent space will not be under any constraints, except the one imposed by the reconstruction loss. We have empirically observed an undesirable behavior where very high standard deviations are output by the encoder. This actually leads to an explosion of the KL loss once this latter is activated after the pre-training. To remove those undesirable effects, a regularization loss $\mathcal{L}_{regstd}$ is added to the original loss defined in Equation 13 such that:

$$\mathcal{L}'_{VGAE}(A, X) = \mathcal{L}_{VGAE}(A, X) + w_{regstd}\mathcal{L}_{regstd}(A, X)$$

$$\text{Where} \quad \mathcal{L}_{regstd}(A, X) = \frac{1}{ND}\sum_{i=1}^{N}\sum_{d=1}^{D}\max[0, \log(\sigma(A, X)_d)]^2$$

In addition to this loss that penalizes high standard deviation, a specific schedule is applied to w_{KL}. Contrary to the cosine annealing used in the β-VAE [28], we will use a cosine-increasing schedule to avoid introducing a weight discontinuity after the pre-training stage. Furthermore, since the KL loss $\mathcal{L}_{KL}$ is limiting the values of the standard deviations, one will no longer need the regularization loss. Thus w_{regstd} will follow a cosine annealing weight schedule.

4.1.4 Regularization of the GMM

The GMM combined with the VGAE also needs to be regularized at some point to avoid undesirable behaviors. Here are the issues that have been faced during the training:

- The variances of the Gaussians are either too large or too small and cause the explosion of the VGAE loss.
- During the training stage, some clusters are becoming orphan clusters, namely no points are belonging to those clusters.

Tackling the first issue is done by clamping the variances of the Gaussian. To do so we introduced boundary constraints in the optimization problem of the EM procedure. The solutions of this new optimization problem are exactly the same as the original one if they are within the boundaries and equal to the boundaries if they are outside them. The choice of the boundaries is done in an iterative way: we start with tight boundaries and we gradually increase them along the training if the standard deviations are still reaching them.

The second issue is actually not a real problem. The fact that a cluster is becoming an orphan cluster may come from different sources, the first one being that we are using too many clusters to model the data. In this latter case, this is not a problem. It becomes an issue as soon as a cluster is becoming an orphan because absorbed by another cluster that now exhibits a bimodal distribution. To prevent this, for every e epochs, if there is a sufficient number of orphan clusters, we will check for each non-orphan cluster that they are not hosting a multi-modal distribution. If it is the case, the following operations are performed:

- The cluster selected is denoted cluster c and has a weight π_c.
- Train a GMM on the data points assigned to the multi-modal cluster. One ends up with two Gaussians parametrized by $\{\pi_0, \mu_0, \sigma_0\}\{\pi_1, \mu_1, \sigma_1\}$.
- Select at random an orphan cluster and assign to this latter the new parameters $\{\pi_0\pi_c, \mu_0, \sigma_0\}$
- Update the parameters of cluster c with the second new parameters $\{\pi_1\pi_c, \mu_1, \sigma_1\}$

Algorithm 2 Training stage of the VGAE and GMM (GMMAE model)

Input:
VGAE parameters, GMM parameters,
K pretain epochs, N epochs,
x model period, y EM period
Initialization:
Initialize the model with VGAE parameters
Train the model for K epochs
Initialize the GMM with GMM parameters
$s = 0$
while $s < N$ **do**
Perform x epochs to train the model
Perform y EM steps to train the GMM
if There are too many orphan clusters **then**
Perform a reassigning step
end if
$s \leftarrow s + x$
end while
Compute cluster labels c_i using Equation 14
Return:
Cluster labels $\{c_1, ..., c_N\}$

The most difficult problem in such an algorithm is to identify whether or not a distribution is multi-modal. To do so one can fit a GMM on the model for a different number of clusters and then use the elbow method to determine if the likelihood increases significantly for more than one cluster. If so the distribution is very likely to be multi-modal. In this work, we did wrong and estimated the multi-modality using the variances of the Gaussian. Our intuition was that there exists a high correlation between large variances and multi-modality, but this is actually false. If you intend to use our algorithms, use the first method. However, this mistake should not hamper our results since replacing one cluster with two clusters is only modifying locally the clustering.

The final training algorithm of the VGAE is given in algorithm 2. The final output of the VGAE that will be leveraged is the cluster labels c_i for each node v_i. This cluster label c_i of a node v_i is computed in the same way it is done for the GMM in Equation 7:

$$c_i = \arg\max_c p(c|\mu_i) = \arg\max_c \frac{p(\mu_i|c)p(c)}{p(\mu_i)} = \arg\max_c \frac{\pi_c \mathcal{N}(\mu_i; \mu_c, \Sigma_c)}{\sum_{j=1}^{C} \pi_j \mathcal{N}(\mu_i; \mu_j, \Sigma_j)} \tag{14}$$

It takes the cluster that has been the more likely to have produce the mean of the node embedding μ_i under the generative process described in definition 12. The mean μ_i is used instead of z_i random vector to ensure a deterministic output.

4.2 Node pooler

Recall that we intend now to merge similar nodes together. The similarity measure used in this work is given by the cluster labels, namely two nodes in the same cluster should be similar in terms of neighborhood. A straightforward way to use the cluster labels would be to merge all the nodes from the same cluster into a Supernode. If one performs this operation, it is very likely that we merge nodes that are very far apart in terms of graph distance, since the nodes' clustering is based on their local structure and completely ignores their betweenness. We then introduce the pooler module on top of the clustering method to preserve graph distances between the nodes. Based on those latter, it will first subdivide the clusters into sub-clusters and then merge all the nodes from the same sub-clusters together. To avoid merging nodes that are too far away from each other, we want our pooler algorithm to satisfy the following constraint:

two nodes can be merged only and only if they are belonging to the same cluster and if their short-path length is smaller than 2

Where the short path length is computed on the undirected version of the graph. Unfortunately, this constraint is not strict enough to uniquely define the sub-clusters (i.e. different sub-clusters may satisfy

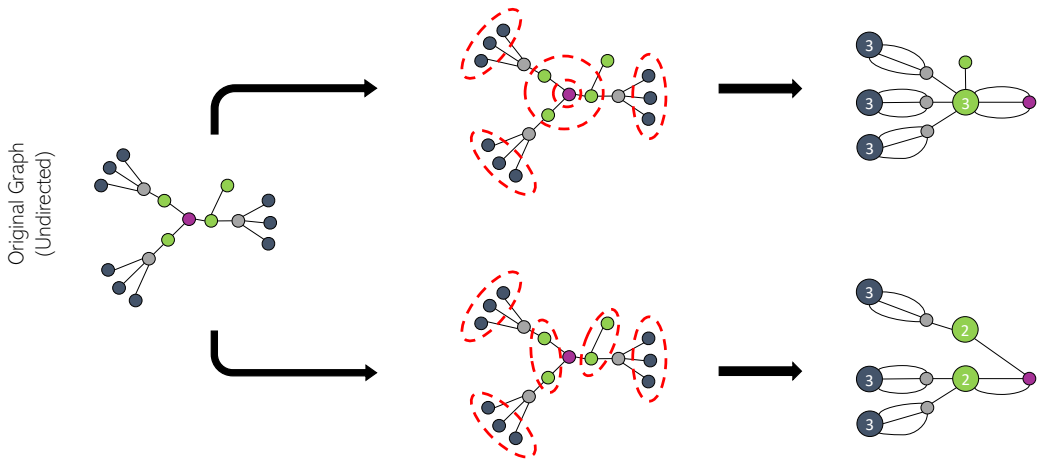

Figure 7: This figure shows, through a toy example, that the initial constraint imposed on the pooler algorithm is not strict enough to ensure the uniqueness of the final sub-clustering. The nodes are colored according to their respective cluster labels. As one can notice, there exist at least two ways to define sub-clusters while satisfying a short-path length smaller than two. One can either group the three green nodes and let one node alone (at the top), or make two groups of two nodes (at the bottom).

this constraint). An example of different sub-clusters satisfying the constraint is shown in Figure 7. To remove this ambiguity we will put an additional constraint on the pooler algorithm. To formulate it, one will first need to define the notion extended adjacency matrix of order k and induced extended graph of order k.

Definition 14. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with N nodes, the extended matrix of order k $A^{(k)} \in \mathbb{R}^{N \times N}$ is defined as follows:*

$$A^{(k)}_{i,j} = \begin{cases} 1 & \text{if } \mathrm{sp}(v_i, v_j) \leq k \\ 0 & \text{otherwise} \end{cases}$$

Where $\mathrm{sp}(v_i, v_j)$ is the short-path length between v_i and v_j. The extended graph of order k $\mathcal{G}^{(k)}(\mathcal{V}, \mathcal{E})$ is the graph induced by $A^{(k)}$. $A^{(k)}$ can also be expressed in terms of matrix multiplication as follows:

$$A^{(k)} = \mathbb{1}\left(\sum_{i=1}^{k} A^k > 0\right)$$

The extended graph of order k consists in connecting each edge to its k-hop-neighborhood. The constraint stated before can then be rewritten using the extended graph of order 2:

two nodes can be merged only and only if they are belonging to the same cluster and if they are connected in $\mathcal{G}^{(2)}(\mathcal{V}, \mathcal{E})$

Thus a group of nodes from the same cluster can be merged into a Supernode only if they are forming a clique in $\mathcal{G}^{(2)}(\mathcal{V}, \mathcal{E})$, where a clique is defined as:

Definition 15. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a subgraph induced by nodes $\{v_{o_1}, ..., v_{o_H}\}$ is forming a clique if and only if:*

$$\forall (i,j) \in \{v_{o_1}, ..., v_{o_H}\} \times \{v_{o_1}, ..., v_{o_H}\} \quad (i,j) \in \mathcal{E}$$

In other words, a subgraph of $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a clique only and only if the subgraph is complete.

The problem of non-uniqueness highlighted previously comes from the fact that a node can belong to different cliques as depicted in Figure 7. To remove the ambiguity, a node will be merged to the biggest clique it belongs to, and if there are more than one biggest cliques (cliques with the same size) then the node will be stated as **uncertain** and remain alone. Let's give a formal definition to biggest node clique and node uncertainty notions:

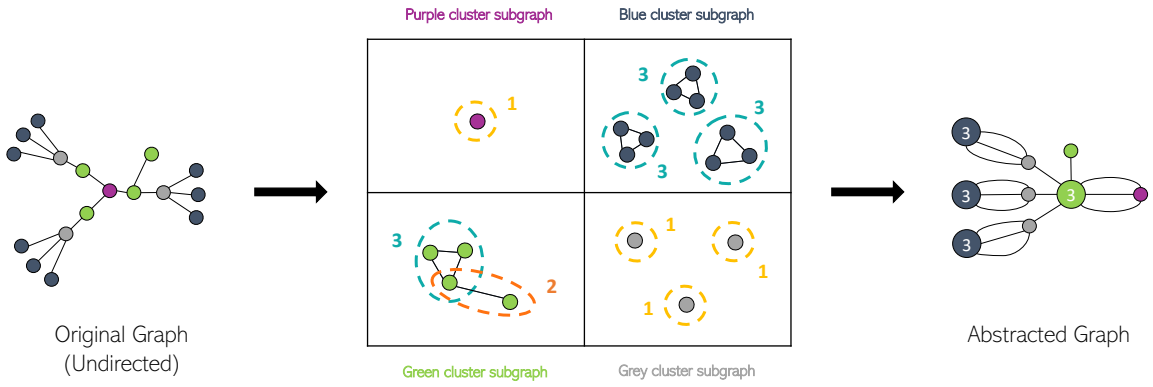

Figure 8: This diagram shows the different steps performed by the pooler algorithm presented in algorithm 3. The nodes are colored according to their respective cluster labels. It starts by building the extended graph of order 2 and extracting the subgraph for each cluster (middle). It then finds all maximal cliques highlighted with dashed circles. Next, it solves the conflict between the different maximal cliques by selecting the biggest ones. In this toy example the clique of size two, in the green cluster subgraph, is discarded in favor of clique of size 3. Finally, it merges the nodes that are in the selected cliques to build a first abstracted graph.

Algorithm 3 Pooler algorithm

Input:
$\mathcal{G}(\mathcal{V}, \mathcal{E})$
Initialization:
Initialize the set of selected cliques $CL_s = \{\}$
Build the extended graph of order 2 $\mathcal{G}^{(2)}(\mathcal{V}, \mathcal{E})$
for each cluster c **do**
Extract the subgraph $\mathcal{G}_c^{(2)}(\mathcal{V}_c, \mathcal{E}_c)$ induced by the nodes $\mathcal{V}_c$ of the cluster c
Initialize the set of remaining nodes $\mathcal{V}_{c,r} = \mathcal{V}_c$
Find all the maximal cliques CL in $\mathcal{G}_c^{(2)}(\mathcal{V}_c, \mathcal{E}_c)$
while $|\mathcal{V}_{c,r}| > 0$ **do**
Select all feasible CL_f cliques in CL
List all the uncertain nodes and store them in a set $\mathcal{V}_{c,u}$
if $|CL_f| > 0$ **then**
Select the biggest feasible clique $bcl = \arg\max_{cl \in CL_f} |cl|$
$CL \leftarrow CL \setminus \{bcl\}$
$CL_s \leftarrow CL_s \cup \{bcl\}$
else
$bcl = \{\}$
end if
for each clique cl in CL **do**
$cl \leftarrow cl \setminus bcl \cup \mathcal{V}_{c,u}$
end for
$\mathcal{V}_{c,r} \leftarrow bcl \cup \mathcal{V}_{c,u}$
end while
end for
Return:
CL_s

Definition 16. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, we denote by $CL_v = \{cl_{v,1}, ...cl_{v,Q}\}$ the set of all the cliques containing v. The biggest clique bcl_v for node v is defined as:*

$$bcl_v = \underset{cl \in CL_v}{\arg\max} |cl|$$

A node is uncertain if and only if:

$$|bcl_v| = |\underset{cl \in CL_v \setminus \{bcl_v\}}{\arg\max} |cl||$$

Concretely we will merge the nodes into their biggest cliques. To satisfy this rule, one must then check two things before merging nodes of a clique: first that all the nodes in the cliques are not uncertain and that the given clique corresponds to the biggest clique for all the nodes into the clique. The clique is then **feasible**. The feasibility of a clique is then defined as:

Definition 17. *A clique cl is feasible if and only if:*

$$\forall v \in cl, \quad cl = bcl_v \quad \text{and} \quad v \text{ is not uncertain}$$

The final pooler algorithm is given in algorithm 3. As one can notice it starts with the biggest feasible cliques and iterates until all the nodes are either merged into a clique or stated as uncertain. Furthermore, as soon as a node is uncertain, it remains uncertain and cannot be merged anymore. All the uncertain nodes will be finally added to the selected cliques CL_s as cliques of size one. This algorithm is illustrated on a toy example on Figure 8. However, this algorithm has the worst disadvantage an algorithm could have, it is **NP-Hard**. The NP-hardness is actually due to the step where the algorithm is searching for the maximal cliques. This is one of Karp's 21 NP-complete problems, and we use the algorithm published by Bron and Kerbosch [29] to solve it. The good point is that this algorithm is not applied to the whole graph, but only to the subgraph extracted from the cluster labels, which is very likely to be disconnected.

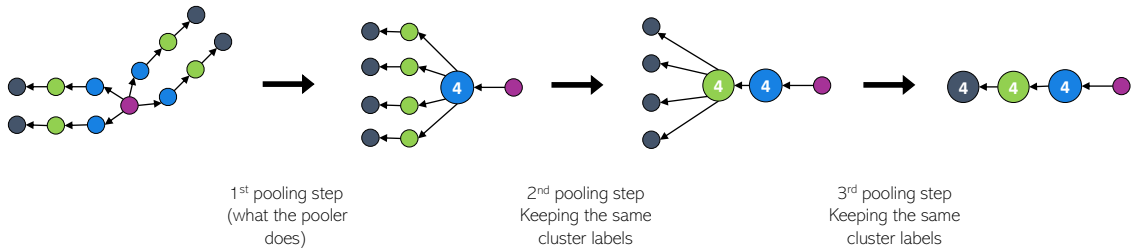

Figure 9: This diagram exhibits how far the cluster labels can be leveraged in an ideal case by simply sequentially reapplying the pooler algorithm several times. The nodes are colored according to their respective cluster labels. Blue nodes are first merged together when applying a first time the pooler algorithm, whereas other nodes remain too distant from one another to be combined. However reapplying a second and a third-time pooler algorithm allows us to merge green and dark blue nodes, resulting in a significantly smaller graph than what we achieved with only one step.

It then remains to define how an abstracted graph is created knowing $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and CL_s. It is performed in the following way:

Definition 18. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$ and a set of non-overlapping cliques $CL_s = \{cl_1, ..., cl_W\}$, the set of supernodes $\mathcal{S_V} = \{s_1, ..., s_W\}$ and superedges $\mathcal{S_E}$ are defined as:*

$$\forall (s_i, r, s_j) \in \mathcal{S_V} \times \mathcal{R} \times \mathcal{S_V}, \quad s_i \neq s_j, \quad (s_i, r, s_j) \text{ appears } n \text{ times in } \mathcal{S_E}$$

$$\text{where} \quad n = |\{(h, r, t) | (h, r, t) \in \mathcal{E}, h \in cl_i, t \in cl_j\}|$$

The label l_{s_i} of a supernode s_i is defined as follows:

$$l_{s_i} = \underset{l \in \mathcal{L}}{\arg\max} |\{v | l_v = l, v \in cl_i\}|$$

The abstracted graph is then defined as the multi directed graph $\mathcal{G}(\mathcal{S_V}, \mathcal{S_E}, \mathcal{R}, \mathcal{L})$.

All the edges are retained in the abstracted graph as soon as its head h and tail t are not in the same Supernode. We are reaching the first corner point of this work: putting the VGAE and the pooler modules together, one is able to generate a first abstracted graph.

4.2.1 Leveraging the cluster labels

The pooler algorithm described in 3 is using cluster labels combined with the graph structure to determine the cliques/supernodes. However, setting that two nodes can be merged only if they have a short path length smaller or equal to 2 is a very restrictive constraint. Finally, one may end up with an abstracted graph where the cluster labels have not been fully leveraged. This intuition is illustrated on Figure 9.

As illustrated in the figure, the application of the pooler algorithm only once results in a significant portion of the nodes remaining unmerged despite having the potential to be merged based on their cluster labels. To fully leverage the output of the VGAE, the superpooler is introduced. It iteratively applies the pooler algorithm until no new supernodes can be created. It will then build a first sequence of Y abstracted graphs $\{\mathcal{G}_{(1,1)}...\mathcal{G}_{(1,Y)}\}$. Those latter will be called low-level abstracted graphs. To avoid a collapse mode, we add an additional rule in the superpooler algorithm, namely that when a supernode with more than two nodes inside is created, it cannot be merged with any other nodes or supernodes in the next iterations. The superpooler algorithm is described in 4. The advantages of the superpooler in comparison to the simple pooler will be extensively discussed in section 7.

4.3 Multi-Level abstraction module

At that point, one is able to generate the first sequence of abstracted graphs by combining the VGAE and the superpooler. The size of the graph will be reduced by a factor that depends on a lot of parameters such as the number of clusters selected, the hyperparameters choice, and mostly the structure of the original graph.

Algorithm 4 Superpooler algorithm

Input:
$\mathcal{G}(\mathcal{V}, \mathcal{E})$, Cluster labels c_i
Initialization:
Initialize the abstracted graph sequence $A_s = \{\mathcal{G}(\mathcal{V}, \mathcal{E})\}$
Set the original graph $\mathcal{O}_g = \mathcal{G}(\mathcal{V}, \mathcal{E})$ and the abstracted graph $\mathcal{A}_g = \emptyset$
Set the sizes of the original and abstracted graph $S_o = |\mathcal{V}|, S_a = -1$
Initialize the set of supernodes with more than one node inside $\mathcal{S} = \{\}$
while $S_o > S_a$ **do**
Compute an abstracted graph $\mathcal{A}_g$, using the pooler 3 and cluster labels $\mathcal{A}_g = \text{pooler}(\mathcal{O}_g, c_i|\mathcal{S})$
Add to $\mathcal{S}$ all the supernodes of $\mathcal{A}_g$ with more than one node inside
$A_s \leftarrow A_s \cup \{\mathcal{A}_g\}$
$S_o \leftarrow |\mathcal{O}_g|, S_a \leftarrow |\mathcal{A}_g|$
$\mathcal{O}_g \leftarrow \mathcal{A}_g$
end while
Return:
A_s

We would like our algorithm to be able to generate abstracted graphs as small as one wants, which is actually not possible with the current configuration.

Our procedure will be improved in a very straightforward way to meet this requirement. The same operations are re-applied on the first abstracted graph generated. In other words, another VGAE will be trained and output a new node partitioning for the abstracted graph, with which the superpooler will generate another sequence of smaller abstracted graphs. This module is called the multi-level abstraction module. To properly define the iterative procedure performed by this module, let's introduce some definitions:

Definition 19. *In the context of the multilevel abstraction module, the abstracted graphs will be indexed by their high level h and their low level l. The high level h of a graph is defined by the number of previous VGAE used to reach this graph. The low level l is defined as the number of times the pooler module has been applied with the same cluster labels to reach this graph. An abstracted graph of high level h and low level l is denoted $\mathcal{G}_{(h,l)}$*

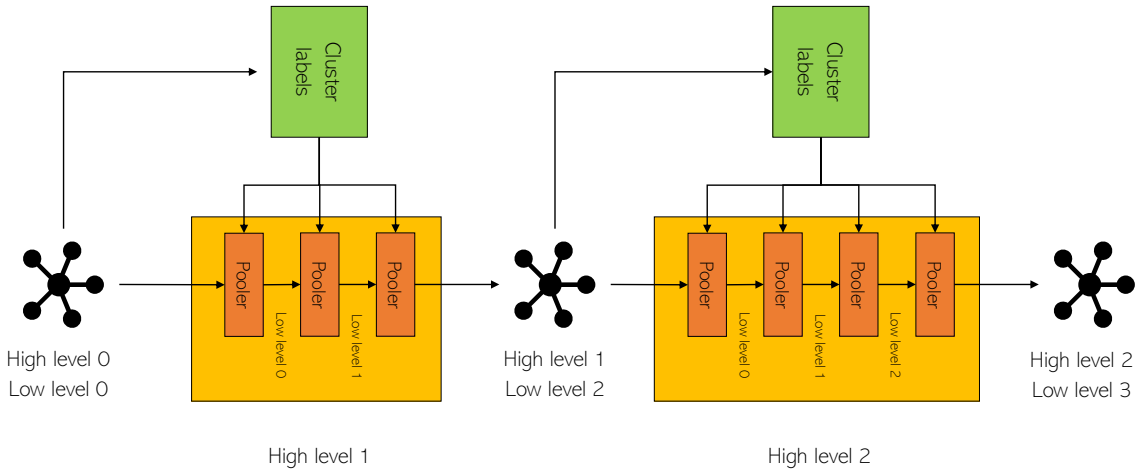

Figure 10: This diagram depicts the way the multi-level abstraction module is building the abstracted graph pyramid. It starts with the original graph (high level and low level equal to 0) and uses a first time a VGAE to generate cluster labels for each node. Based on those cluster labels, the pooler algorithm is applied several times, until no more nodes can be merged. This way, low-level graphs of high-level 1 are generated. The highest low-level reached for high level 1 is 2 in this toy example. Then another VGAE is trained on this latter to generate new cluster labels with which the superpooler algorithm will build another sequence of low-level abstracted graphs.

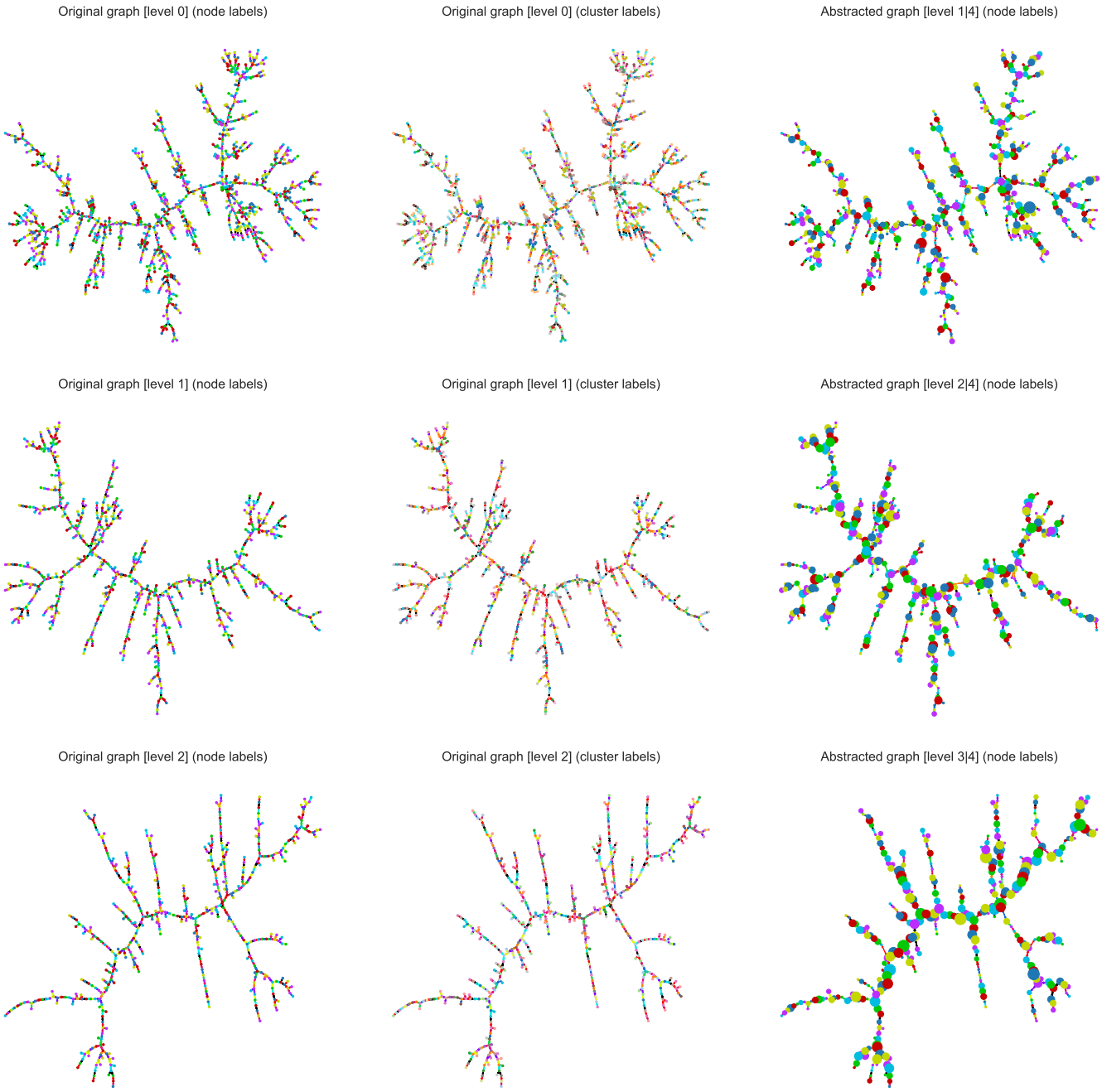

Figure 11: Those figures show the performances of the multi-level abstraction module on a random tree synthetic graph. The different rows correspond to the different high level of abstraction. The first column depicts, for each high level, the graph before clustering and pooling operations are performed. The nodes are colored according to their respective node labels. The second column shows the graph relabeled with the cluster labels generated with the VGAE. The third column shows the abstracted graph made of supernodes, obtained after the pooling operation. The supernode sizes represent the number of nodes that are contained into the supernodes. The highest low level reached for each high level is specified in the title of the right column figures. It is actually equal to 4 for every high level.

Algorithm 5 Multi-level abstraction algorithm

Input:
$\mathcal{G}(\mathcal{V}, \mathcal{E})$, Cluster labels c_i, number of high levels H
Initialization:
Initialize the abstracted graph sequence $A_s = \{\mathcal{G}(\mathcal{V}, \mathcal{E})\}$
Set the original graph $\mathcal{O}_g = \mathcal{G}(\mathcal{V}, \mathcal{E})$ and the abstracted graph $\mathcal{A}_g = \emptyset$
$h = 1$
while $h \leq H$ **do**
Train a VGAE using algorithm Algorithm 2 on $\mathcal{O}_g$ and get cluster labels c_i
Get the sequence of low-level abstracted graphs using the superpooler $A_s^{(h)} = \{\mathcal{G}_{(h,1)}...\mathcal{G}_{(h,Y_h)}\}$
$A_s \leftarrow A_s \cup A_s^{(h)}$, $\mathcal{O}_g \leftarrow \mathcal{G}_{(h,Y_h)}$
$h \leftarrow h + 1$
end while
Return:
A_s

The notion of high level and low level is further illustrated in Figure 10. The multi-abstraction module can be summarized in the algorithm 5, and the results of such a module on a real graph are exhibited in Figure 11.

4.4 Scaling up

The last point tackled in this section is related to the scalability of the methods presented previously. This section is not complete and mainly addresses the problem of VGAE training for large graphs. Indeed the VGAE requires the whole graph to fit into the memory for the training stage. Since GPUs are used to significantly speed up the training, it will require having GPUs with a large amount of memory, devices that are quite expensive. This section introduces a technique that involves providing the VGAE with carefully selected subgraphs, while ensuring that the embeddings produced by the VGAE are identical to those that would be generated using the entire graph as input. The core of this method relies on a specific property of the relational graph convolution operator, called the **message passing trick**.

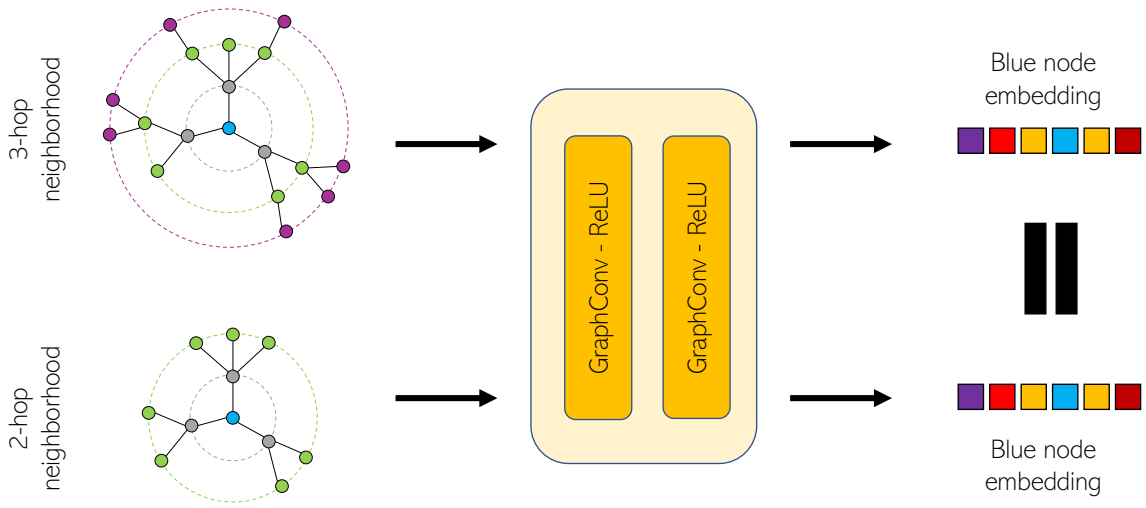

Figure 12: This figure depicts the message passing principle, namely that with a two layers GCN, only the 2-hop neighborhood of a node is needed to compute its embedding. In this toy example, one intend to compute the embedding of the blue node in the middle. Then the blue node embeddings computed using the 3-hop neighborhood (at the top) and the blue node embeddings using the 2-hop neighborhood (at the bottom) are rigorously equal.

4.4.1 Message passing trick

The message-passing trick states that:

Theorem 3. *Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, and a RGCN with k layers where BatchNorm is not used, the embeddings of a node v can be computed using its k-hop neighborhood $\mathcal{G}(\mathcal{V}_{v,k}, \mathcal{E}_{v,k}, \mathcal{R}, \mathcal{L})$ only:*

$$\text{RGCN}_k[h_v|\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})] = \text{RGCN}_k[h_v|\mathcal{G}(\mathcal{V}_{v,k}, \mathcal{E}_{v,k}, \mathcal{R}, \mathcal{L})]$$

Note that this result is still valid for GCN as well. Stated differently, this property implies that computing the embedding of a node only requires information up to its k-hop neighborhood. The message passing trick is illustrated in Figure 12. Thus a naive way to compute the embeddings of all the nodes without feeding the VGAE with the whole graph would be to select for each node its k-hop neighborhood and compute its embedding. It will, unfortunately, leads to a high-time complexity since you will need to run the forward pass as many times as you have nodes to just perform one epoch.

One has to find a middle ground between using the whole graph and using each node separately. To better formulate the problem let's introduce the concept of useful and useless nodes.

Definition 20. *Given a subgraph $\mathcal{G}_\mathcal{S}(\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S}, \mathcal{R}, \mathcal{L})$ of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{L})$, a node $v \in \mathcal{V}_\mathcal{S}$ is useless if:*

$$\mathcal{G}(\mathcal{V}_{v,k}, \mathcal{E}_{v,k}, \mathcal{R}, \mathcal{L}) \nsubseteq \mathcal{G}_\mathcal{S}(\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S}, \mathcal{R}, \mathcal{L})$$

And node v is useful if:

$$\mathcal{G}(\mathcal{V}_{v,k}, \mathcal{E}_{v,k}, \mathcal{R}, \mathcal{L}) \subseteq \mathcal{G}_\mathcal{S}(\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S}, \mathcal{R}, \mathcal{L})$$

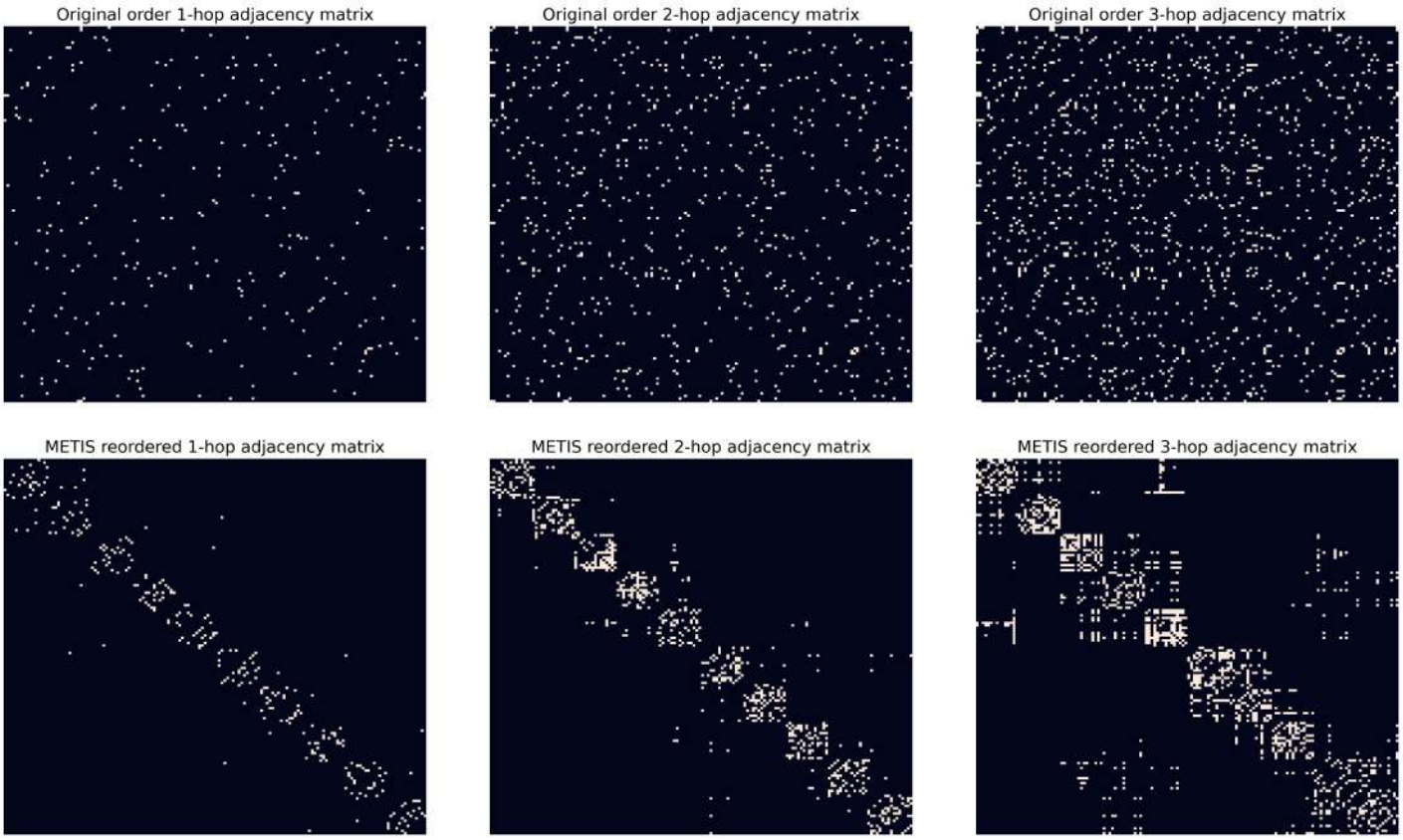

Figure 13: Those figures show METIS performances on a random tree synthetic graph. The extended adjacency matrices of different orders (1,2 and 3) are represented in the first row without performing any reordering operation. The second row shows the exact same adjacency matrix where the node indexes have been reordered using METIS partitioning with $G = 10$. The partitioning provided by METIS allows to block-diagonalize the adjacency matrix.

Algorithm 6 Subgraph generation

Input:
$\mathcal{G}(\mathcal{V}, \mathcal{E})$, number of groups G, number of layers of the RGCN k
Initialization:
Initialize the set of subgraphs $\mathcal{S} = \{\}$
Partition graph nodes into G groups, $\{\mathcal{V}_{\mathcal{S}_1} ... \mathcal{V}_{\mathcal{S}_G}\}$ using METIS
for each $\mathcal{V}_{\mathcal{S}_i}$ **do**
$\hat{\mathcal{V}}_{\mathcal{S}_i} = \{\}$
for each node v in $\mathcal{V}_{\mathcal{S}_i}$ **do**
$\hat{\mathcal{V}}_{\mathcal{S}_i} \leftarrow \hat{\mathcal{V}}_{\mathcal{S}_i} \cup \mathcal{V}_{v,k}$
end for
Extract the subgraph $\mathcal{G}_{\mathcal{S}}$ induced by $\hat{\mathcal{V}}_{\mathcal{S}_i}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{G}_{\mathcal{S}}\}$
end for
Return:
$\mathcal{S}$

In other words a node of a subgraph is useful if its k-hop neighborhood is included in the subgraph. The objective is to create subgraphs that are of limited size and quantity while ensuring that every node is useful in at least one of the subgraphs. To do so, we will follow intuition from Cluster-GCN [19] and use METIS algorithm [30].

4.4.2 METIS partitioning

METIS is a collection of serial programs designed for graph partitioning, finite element mesh partitioning, and generating fill-reducing orderings for sparse matrices. Its algorithms are based on the multilevel recursive bisection, multilevel k-way, and multi-constraint partitioning schemes. As noted in Cluster-GCN [19], METIS can create partitions that maximize within-cluster links and minimize between-cluster links. Regarding the adjacency matrix, METIS tries to block diagonalize it into G groups where G is an input parameter of METIS.

Cluster-GCN [19] applies METIS on the adjacency matrix A and discards off-diagonal edges of the adjacency matrix to generate the subgraphs. This way they only get an approximation of the embeddings. In this work, we aim to compute the exact value of the embeddings, thus one needs to integrate into the subgraphs the k-hop neighborhood of each node. To group nodes with high overlapping k-hop neighborhoods, we rather apply the METIS algorithm on the extended matrix of order k, namely $A^{(k)}$. An example of the performances provided by METIS is exhibited in Figure 13. Then for each subgraph $\mathcal{G}_{\mathcal{S}}$ found by METIS, we add all the necessary nodes to build $\hat{\mathcal{G}}_{\mathcal{S}}$ such that it satisfies:

$$\forall v \in \mathcal{V}_{\mathcal{S}}, \quad v \text{ is a useful node with respect to } \hat{\mathcal{G}}_{\mathcal{S}}$$

In other words, we add the k-hop neighborhood of all the nodes of $\mathcal{G}_{\mathcal{S}}$. Since the node sets of the subgraphs found by METIS are forming a partition of the nodes of the original graph, for each node $v \in \mathcal{E}$ it exists a subgraph in which node v is useful. The size of the subgraphs generated as well as the scaling-up potentialities will be deeply analyzed in section 6. The algorithm applied on the top of METIS is described in algorithm 6.

5 Experimental set-up

The methods presented previously are assessed on two datasets, namely, a SwissCom Monty (SCM) dataset, and the Word Net graph denoted WNS [**?**]. The GMMAE model, which employs VGAE in conjunction with GMM, will be compared to two additional models:

- A **Baseline model** for which the cluster labels c_i are directly inferred from the labels l_i. This model will be evaluated only on subgraphs since it has an intractable time complexity on the full graph[7].
- A **Graph Auto-Encoder (RGAE)** having the same architecture presented in Figure 5 and where the cluster labels c_i will be inferred from a GMM that will be applied on the latent space of the trained RGAE.

To evaluate the performances of the models for different graph sizes, the methods will be assessed on the full graphs but also on subgraphs of smaller sizes. Those pre-processing steps are detailed in subsection 5.1. The parameters used for the GMMAE model and the RGAE model are presented in subsection 5.2 and in subsection 5.3. The metrics used to compare the different methods are finally described in subsection 5.4.

5.1 Dataset and pre-processing

As stated in the previous sections, the graphs are multi-directed, meaning that an edge with the same head, relation, and tail may exist more than one time. To be able to leverage Theorem 1 the relations are inverted such that for each relation r, an inverse relation r' is created and each edge (h, r, t) is doubled with an edge (t, r', h). For the node features, the label matrix $\bar{L}$, defined in Equation 1, is used.

Subgraphs of different sizes are generated from the whole graph to assess the behavior of the model on graphs with different sizes. Regarding SCM, the data set was already split into subgraphs of small sizes, and subgraphs of higher sizes are generated by sequentially merging the subgraphs together. Regarding WNS data set, we used the METIS algorithm on the original adjacency matrix to preserve as many edges as possible and discarded off-diagonal elements. For SCM data set, graphs of size 500, 2000, 5000, and 10000 are extracted from the full graph while for the WNS data set, graphs of size 2000, 5000, and 10000 are extracted from the full graph.

To evaluate the generalization capabilities of our methods, we split for each subgraph category the data set into a train and a test set. Regarding the full graph, the methods will be assessed directly on it. The influence of some hyperparameters of the models will be assessed using K-fold cross-validation on the train set. The train and test set sizes, as well as basic descriptions of the graph datasets, is reported in Table 1:

Dataset	Size		$N = \lvert\mathcal{V}\rvert$		$E = \lvert\mathcal{E}\rvert$		R	L
	train	test	train	test	train	test		
SCM500	30	12	0.5	0.5	1.4	1.3	24	6
SCM2K	18	7	2.0	2.0	5.8	5.9	24	6
SCM5K	10	3	5.0	5.0	15.2	15.2	24	6
SCM10K	5	2	10.0	10.0	31.0	31.1	24	6
SCMfull	1	_	70.2	_	223.9	_	24	6
WNS2K	15	5	2.0	2.0	7.3	7.8	22	5
WNS5K	6	2	5.1	5.1	18.9	21.3	22	5
WNS10K	3	1	10.1	10.1	39.1	43.7	22	5
WNSfull	1	_	40.6	_	173.5	_	22	5

Table 1: The characteristic for each data set are specified in the above table. The number of nodes N and edges E are computed using the average over the training and test set. R represents the number of relations after the inversion step, and L the number of node labels.

[7] This is mainly due to the poor cluster assignment provided by the baseline model, which increases the time complexity of the pooler algorithm.

Dataset	Encoder dim		Latent dim	Decoder layers		Components	Params (K)	batch size	epochs
	GCN	MLP		Adjacency	Label				
SCM500	64	48	32	3	2	25-22-20	264	6	300
SCM2K	64	48	32	3	2	30-26-22	264	4	300
SCM5K	96	64	48	3	2	35-30-25	575	2	300
SCM10K	96	64	48	3	2	35-32-30	576	2	300
SCMfull	128	80	48	3	2	50-45-40	997	1	600
WNS2K	96	64	48	4	3	30-27-25	532	3	350
WNS5K	96	64	48	4	3	40-35-30	532	2	350
WNS10K	128	80	48	4	3	40-35-30	918	1	350
WNSfull	128	80	48	4	3	50-45-40	920	1	600

Table 2: The hyperparameters of the VGAE architecture as well as the training parameters are reported in the above table. In addition, 3 layers were used both for the RCGN and the upstream MLP of encoder. The Components entry corresponds to the number of initial clusters for the GMM for each high level.

5.2 VGAE configuration

The VGAE is trained using Adam with an initial learning rate $\eta = 0.01$ and momentum $\beta = (0.9, 0.999)$. A cosine-annealing schedule with lower-bound $\eta_{min} = 5 \times 10^{-4}$ is applied to the learning rate. The number of positive edges sampled is actually equal to the whole set of edges $\mathcal{E}_{pos} = \mathcal{E}$ since the average degree is small enough. The number of negative edges sampled is set to $\mathcal{E}_{neg} = 3\mathcal{E}$ in those experiments. The logarithm of the standard deviation of the encoder is clamped at 3 for stability purposes. For SCM data set we used the following weights in the loss: $w_{KL} = 0.3$, $w_{relations} = 1.3$, $w_{labels} = 0.8$ and $w_{regstd} = 0.1$ while for WNS the following ones are used $w_{KL} = 0.6$, $w_{relations} = 1.5$, $w_{labels} = 1.2$ and $w_{regstd} = 0.3$. As specified before, a cosine-increasing schedule is applied to w_{KL} starting from 0, and a cosine-annealing schedule is applied to w_{regstd} with a zero lower bound. The other training parameters and hyperparameters used for the architecture are reported in Table 2.

5.3 GMM configuration

The Gaussian Mixture Model is initialized 100 epochs, respectively 125 epochs, after the beginning of the training for the SCM data set, respectively WNS data set. It performs a maximum of 50 EM steps every 25 epochs for the subgraphs and 50 epochs for the full graphs. The algorithm performs cluster reassignment when the number of orphan clusters exceeds one-quarter of the initially assigned clusters. The reassignment starts at 35 % of the training stage, with a period of 50 epochs and patience of 2. A last reassigning step is performed at 70 % of the training stage. The logarithm standard deviations of the GMM are initially clamped between -0.7 and 0.6 and extended if reached. The update is done each EM step with patience of 4.

5.4 Evaluation stage

The primary issue with this project is the absence of metrics to evaluate the performance of the presented algorithms. In order to address this, we have devised "unsupervised metrics" to first ensure compliance with the initial requirements specified in section 2, and secondly, to examine other metrics that can indicate the quality and consistency of the clustering. Although the metrics will be defined at the graph level, the test set comprises multiple graphs. Hence, the metrics presented in section 6 will be computed by taking the mean over the graphs in the test set.

5.4.1 Auto-Encoder performances

To assess the performances of the auto-encoder in the reconstruction of the adjacency matrix, two common metrics are used, namely the Area Under the Curve (AUC) and the Average Precision score (AP). For a given set of positive edges $\mathcal{E}true, pos$ and negative edges $\mathcal{E}true, neg$, the Auto-Encoder produces a probability of edge existence $p(i, j)$ for each edge. Putting a threshold t on the probabilities, one is able to compute the false positive rate $FPR(t)$ and the $TPR(t)$ of such a classifier, FPR, and TPR are defined as follows:

$$FPR = \frac{FP}{N} \qquad TPR = \frac{TP}{P}$$

Where FP (False positive) are the number of edges in $\mathcal{E}_{true,neg}$ predicted as positive (existing) by the model, $N = |\mathcal{E}_{true,neg}|$, TP (True positive) are the number of edges in $\mathcal{E}_{true,pos}$ predicted as positive (existing) by the model and $P = |\mathcal{E}_{true,pos}|$. TPR is also commonly called the Recall metric. The AUC is defined as follows:

$$\text{AUC} = \int_0^1 TPR(FPR^{-1}(t))dt \tag{15}$$

Assuming $FPR(t)$ is invertible. In other words, it is the area under the receiving operator curve. The Average Precision AP is very similar to the AUC but computes the area under the precision Pre, recall Rec curve where the precision is defined as:

$$Pre = \frac{TP}{FP + TP}$$

The AP metric, assuming the recall function is invertible, is computed as follows:

$$\text{AP} = \int_0^1 Pre(Rec^{-1}(t))dt \tag{16}$$

Since the integrals cannot be computed, they are approximated using a finite set of thresholds and either the trapezoidal or rectangle rule. Furthermore, we set $\mathcal{E}_{true,pos} = \mathcal{E}$ and sample 5 different sets of negative edges $\mathcal{E}_{true,neg}$, computing the mean AUC and AP across these 5 samples.

5.4.2 Distance correlation

Recall that our multi-level abstraction module intends to preserve the distances between two nodes between two successive abstracted graphs. To assess if it satisfies such a requirement, we will compute the correlation coefficient between two wisely chosen random variables. Let's consider a high level $h > 0$ and the sequence of abstracted graphs generated by the superpooler $\{\mathcal{G}_{(h,1)}...\mathcal{G}_{(h,Y_h)}\}$. We consider two successive abstracted graphs $\mathcal{G}_{(h,l)}$ and $\mathcal{G}_{(h,l+1)}$. Now let's choose at random two nodes in $\mathcal{G}_{(h,l)}$ denoted U and V and we define the following random variable:

$$X_{(h,l)}(U,V) = \text{sp}_{\mathcal{G}_{(h,l)}}(U,V) \quad \text{with} \quad U \sim \mathcal{U}(\mathcal{V}_{h,l}) \quad V \sim \mathcal{U}(\mathcal{V}_{h,l})$$

Where $\text{sp}_{\mathcal{G}}(U,V)$ is the short path length between U and V in graph $\mathcal{G}$. To assess if the graph distances are preserved, we compute the correlation coefficient between $X_{(h,l)}(U,V)$ and $X_{(h,l+1)}(S_U, S_V)$ where S_U, respectively S_V, is the supernode in $\mathcal{G}_{(h,l+1)}$ containing U, respectively V. In other words, one computes:

$$\rho_{(h,l)} = \frac{\text{Cov}[X_{(h,l)}(U,V), X_{(h,l+1)}(S_U,S_V)]}{\sigma_{X_{(h,l)}(U,V)}\sigma_{X_{(h,l+1)}(S_U,S_V)}}$$

If $\rho_{(h,l)}$ is close to 1, it qualitatively means that if a node u and a node v are far away from each other in $\mathcal{G}_{(h,l)}$, they are still relatively far away in $\mathcal{G}_{(h,l+1)}$. We aggregate the $\rho_{(h,l)}$ for a high level h using a weighted sum where the weights correspond to the difference in the number of nodes between $\mathcal{G}_{(h,l)}$ and $\mathcal{G}_{(h,l+1)}$. The final metric $\hat{\rho}_{(h)}$ we compute for a given level h is computed as follows:

$$\hat{\rho}_{(h)} = \frac{\sum_{l=-1}^{Y_h-1} \rho_{(h,l)} g_l}{\sum_{l=-1}^{Y_h-1} g_l} \quad \text{with} \quad g_l = |\mathcal{V}_{h,l}| - |\mathcal{V}_{h,l+1}| \tag{17}$$

With the convention that $\mathcal{G}_{(h,-1)} = \mathcal{G}_{(h-1,Y_{h-1})}$. To estimate $\rho_{(h,l)}$, 5000 pairs of nodes are sampled 5 times and we take the mean over the 5 repetitions to estimate $\rho_{(h,l)}$.

5.4.3 Homophilies

To evaluate if requirement 2 is satisfied we will measure the neighborhood homophily into clusters. For the sake of simplicity, we will relax the definition of the neighborhood of a node given in definition 13. Given a node v its neighborhood is defined by the label distributions of the nodes around him, it then can be mathematically represented by the following vector:

$$x_v = \frac{1}{\mathcal{N}(v)} \sum_{u \in \mathcal{N}(v)} \bar{L}_u$$

Where $\bar{L}_u$ is the row in the label matrix corresponding to the node u. Then given a cluster c containing nodes $\mathcal{V}_c$, one is able to evaluate how far the neighborhood of the nodes are similar computing the following vector:

$$\sigma^2_{neig,c} = \frac{1}{|\mathcal{V}_c|} \sum_{v \in \mathcal{V}_c} (x_v - \bar{x}_c)^2 \quad \text{with} \quad \bar{x}_c = \frac{1}{|\mathcal{V}_c|} \sum_{v \in \mathcal{V}_c} x_v$$

It is actually no more no less than the standard deviation estimator of the x_v. To aggregate the clusters, we use a weighted average sum, where clusters with a larger number of points are given greater weight than those with fewer points. The final metric called mean neighborhood standard deviation and denoted μ_σ is computed as follows:

$$\mu_\sigma = \frac{\sum_{c=1}^{C} |\mathcal{V}_c| \bar{\sigma}_{neig,c}}{\sum_{c=1}^{C} |\mathcal{V}_c|} \tag{18}$$

Where $\bar{\sigma}_{neig,c}$ is the mean of the vector $\sigma_{neig,c}$. Thus the lower the mean neighborhood standard deviation the higher the neighborhood homophily is. In addition, we will also record the label homophily inside the clusters. A high label homophily means that the nodes that are in the same cluster have the same label. One can compute the **Gini** impurity on the label distribution of a cluster c to estimate the label homophily:

$$\text{Gini}_c = \frac{L-1}{L} \sum_{i=1}^{L} p_{c,i}(1 - p_{c,i}) \quad \text{with} \quad p_{c,i} = \frac{1}{|\mathcal{V}_c|} \sum_{v \in \mathcal{V}_c} \bar{L}_{v,i}$$

With L the number of labels. A **Gini** impurity of 0 means that the distribution is pure meaning $\exists\ i$ s.t $p_{c,i} = 1, \forall j \neq i \quad p_{c,j} = 0$ while a **Gini** impurity of 1 means that the distribution is uniform. Thus the lower the **Gini** impurity the higher the label homophily is. The impurities of all the clusters are again aggregated using a weighted average such that:

$$\text{Gini} = \frac{\sum_{c=1}^{C} |\mathcal{V}_c| \text{Gini}_c}{\sum_{c=1}^{C} |\mathcal{V}_c|} \tag{19}$$

5.4.4 Silhouette score

To assess how far the embeddings are exhibiting a cluster-friendly distribution, the silhouette score well-known metric is. Given a node v belonging to cluster c, its silhouette score s_v is defined as:

$$\begin{aligned} s_v &= \frac{a(v) - b(v)}{\max(a(v), b(v))} \\ a(v) &= \frac{1}{|\mathcal{V}_c| - 1} \sum_{u \in \mathcal{V}_c} \| z_u - z_v \|^2 \\ b(v) &= \min_{j \neq c} \frac{1}{|\mathcal{V}_j|} \sum_{u \in \mathcal{V}_j} \| z_u - z_v \|^2 \end{aligned} \tag{20}$$

Where z_v is the embedding vector associated with node v. The total silhouette score is computed by computing the mean over all the nodes.

5.4.5 Compression factor

To assess how far the methods can compress the graph we will record the compression factor between two high levels h and $h+1$ as follows:

$$f_c = \frac{|\mathcal{V}_{h,Y_h}|}{|\mathcal{V}_{h+1,Y_{h+1}}|} \tag{21}$$

In other words, it computes by which factor the size of the node set has been divided from one high level to another.

5.4.6 Robustness Metrics

The last metric that will be recorded intends to measure the consistency of our algorithms, especially the consistency of our node clustering. To put it another way, the goal is to quantify the similarity between two node clustering produced by two identical models that were initialized with different seeds. The **V-measure** has been specifically developed to fit this need in [31]. This function accepts two cluster assignment vectors and outputs a scalar value between 0 and 1. A value of 1 indicates that the clustering assignments are completely identical, while a value of 0 indicates that the clustering assignments are completely dissimilar. We will use the V-measure on two cluster assignments:

- We will compute the V-measure on the cluster labels assigned to each node just before pooling the nodes. We will call this metric, the clustering V-measure.
- We will compute the V-measure based on the supernode generated after the pooling operation. Namely nodes that are pooled into the same supernode have the same cluster label. We will call this metric, the pooling V-measure.

6 Results

The following results were obtained using a Tesla V100 GPU. The methods described in section 4 were implemented using the pytorch geometric [32] and networkX [33] libraries. The code is made available at this github.

Three different models will be compared in this section, namely the baseline model, RGAE and GMMAE. The only part that differs from one model to another is the way the cluster labels are computed. In this section, we will first take a closer look at some specific part of the GMMAE model. The performances of the superpooler against the one of the pooler are shown in subsection 6.1 while the influence of the number of initial clusters used to model the latent space is discussed in subsection 6.2. Then, the three models will be compared based on their overall performances in subsection 6.3 and their robustness subsection 6.4. Finally we will assess to what extent the scaling-up methods subsection 4.4 can be applied to the SCM data sets.

6.1 Benefits of the superpooler

The superpooler was introduced to replace the pooler in order to leverage the cluster labels. Its objective is then to increase the compression factor while not hampering the other metrics, such as the the correlation coefficient and especially time performances. By definition the time performances of the pooler will be better than the superpooler ones (since it applies several times the pooler module). However one could hope that the time it takes to perform the pooling operation can be neglected compared to the time it takes to train the VGAE, such that the compression factor will be increased at a smaller cost.

To confirm our intuition, experiments were run on the different subgraphs of SCM. The GMMAE model is used and only the first high level is computed. The results were obtained using a K-fold cross validation on the train set with $K = 5$, the test set is then ignored since the superpooler can be considered as an hyperparameter. The results are reported in Figure 14.

Unsurprisingly, the compression factor is highly increased with the introduction of the superpooler (by a factor 2 at least and a factor 6 at most). This is a very good property since reducing the size of the graph with the same cluster labels will also contribute to speed up the training of the VGAE of the next level of abstraction. Regarding the mean correlation coefficient, the pooler achieves slightly better performances than the superpooler does. This latter still achieve good results with a lower bound of 0.88. Finally, in terms of time performances, as expected the pooler is faster than the superpooler. Nevertheless the time difference can be neglected compared to the VGAE training time. Furthermore, it is reasonable to anticipate that the time invested in re-executing pooling operations will yield greater time savings by accelerating the VGAE training for subsequent levels of abstraction.

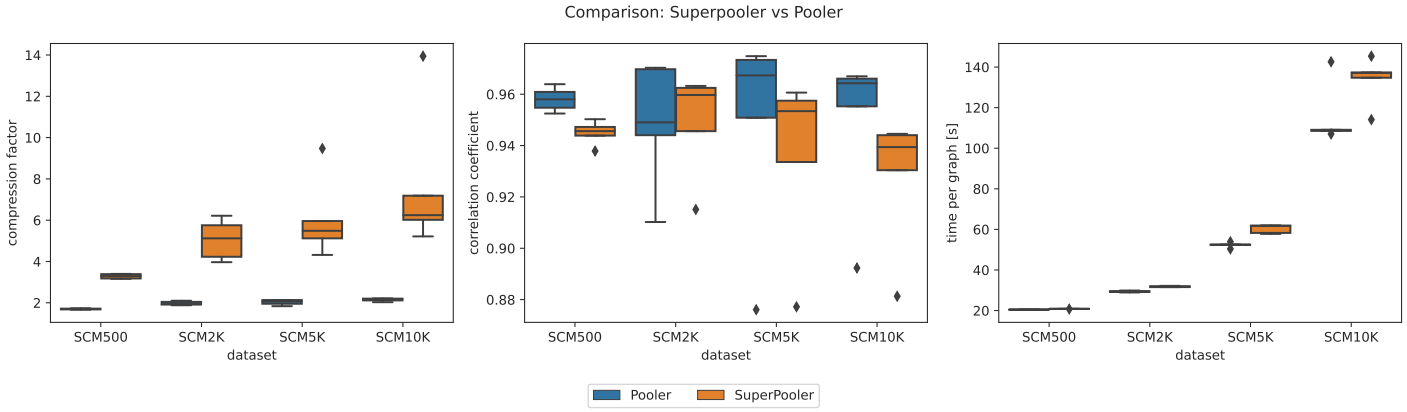

Figure 14: Those experiments aim to compare the performances of the SuperPooler against the Pooler. From left to right: compression factor defined in Equation 21, correlation coefficient defined in Equation 17, and time performances. GMMAE model is used to generate the cluster labels. The results presented above were obtained using K-fold cross-validation with $K = 5$ on the training set of the different data sets. The metrics are computed on the validation set.

6.2 Initial number of clusters

We will now explore a second important aspect of the GMMAE model namely the influence of the number of initial clusters. The first natural question that arises is how to set the number of cluster of the GMM ? If one chooses a low number of clusters, there is a risk that we underfit the distribution of the latent space. On the contrary, one risks to overfit this latter if the number of clusters is too high. Well, the results will in some sense contradict this last statement, and suggests to put more initial clusters than one would think there are.

In those experiments, the compression factor, the homophilies, the silhouette score as well as the Auto-Encoder performances are recorded for different number of clusters. In addition, a specific metric we call the number of remaining Gaussians will also be recorded. To put it simply, this metric computes, at the end of the training stage, the number of Gaussians for which at least one point is assigned to it. Then the difference between the initial number of Gaussians and the remaining ones is equal to the number of Gaussians that are not used by the model, they are, in other words, kind of useless. Similarly to the previous experiments, those ones were run on the different subgraphs of SCM. The GMMAE model is used and only the first high level is computed. The results were obtained using a K-fold cross validation on the train set with $K = 5$. They are depicted in Figure 15.

Regarding the homophily metrics, specifically mean impurity and mean neighborhood standard deviation, it was anticipated that these metrics would decrease. Indeed a higher number of clusters will give to the model a higher granularity such that it will be able to separate node neighborhoods that differs only from a few quantity. The increasing behavior of the AUC, AP and silhouette score could have also been expected. More specifically, the fact that they are low for a low number of clusters reflect the fact that we underfit the distribution of the latent space. Surprisingly the compression factor oscillates with low amplitude around a constant, while one could have expected that the compression would have decreased as the number of clusters is increasing. Such an oscillation may suggest that nodes with different neighborhoods are far away from each others in the graph.

The last, but no least, metric to analyze is the number of remaining Gaussians. For a low number of initial Gaussians, one can notice that the curves are sticking to the triangular curve which means that all the Gaussians are used. As the number of Gaussians increased, the curves drop to stabilize to a constant. This strange behavior can be explained by the fact that the model only keeps

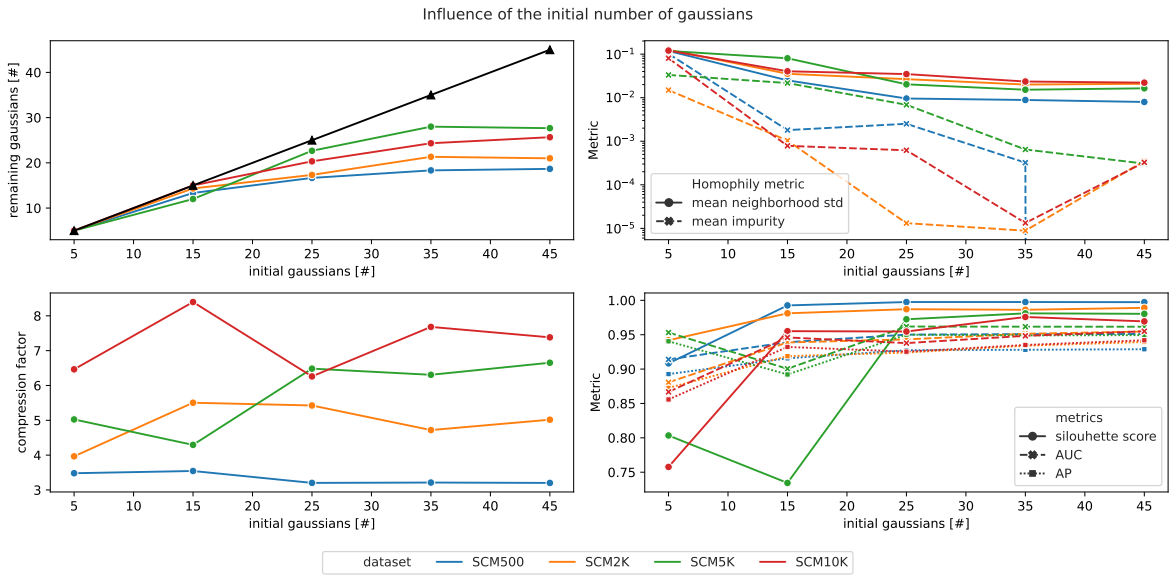

Figure 15: Those experiments aim to assess the sensitivity of the GMMAE model with respect to the initial number of clusters. From left to right, top to bottom: number of remaining Gaussians, homophilies defined in Equation 19 and Equation 18, compression factor defined in Equation 21, silhouette score, AUC and AP respectively defined in Equation 20 Equation 15 and Equation 16. The results presented above were obtained using K-fold cross-validation with $K = 5$ on the training set of the different data sets. The metrics are computed on the validation set.

the necessary clusters and discards the useless ones. It first avoids overfitting the data distribution and secondly it makes easier the choice of the number of Gaussians C, since setting a higher value than the optimal one does not, a priori, impact the performances. However it has been empirically shown that setting C too high may lead to instabilities in the training processes of both the GMM and VGAE, thus we do not recommend to set C to a very high value (¿ 100).

6.3 Overall performances

The baseline, RGAE and GMMAE models will be compared using the sets of metrics described in subsection 5.4, namely the homophily metrics, the correlation coefficient, the compression factor, the silhouette score, the Auto-Encoder metrics and finally the time performances. Those metrics are computed on the test set (except for the full graphs) and 5 repetitions with different seeds (1 repetition for the baseline since it is deterministic) are performed to estimate them. The multi-level abstraction module is used with 3 high levels and the metrics are computed for each level. We first analyze the results on SCM data sets and then the results on WNS.

6.3.1 SCM

To first check how far requirement 2 is satisfied, we will observe how the models perform in terms of homophily metrics on SCM. The results are depicted on Figure 16. Regarding the mean impurity, the first thing that one can notice is that the baseline model is performing better than all the other ones but this is not a surprise since by definition the cluster labels are inferred from the node labels. GMMAE model performs well in terms of mean impurity, reaching almost 0, for each high level, while RGAE model tends to group nodes with different labels together which leads to a high mean impurity. RGAE model reaches mean impurities greater than 0.1 in SC10K and SC500. Before going further, one should explain why having impure clusters reflects bad performances in the special case of SCM. The structure of SCM is such that nodes with similar labels are very likely to share similar local structures. We do not say that all the nodes having the same labels are sharing the same local structure (in which case the baseline model would outperform the other models), we rather say that there exist subgroups of nodes having the same node labels that are sharing similar local structures. Thus if one intends to group nodes with similar local structures, the groups should be pure in terms of node labels. To verify to what extent the methods are grouping nodes with a similar local structure, one can look at the mean neighborhood standard deviation. Unsurprisingly, the baseline model is out of the race since it totally ignores the neighborhood of the nodes to cluster them. The RGAE and

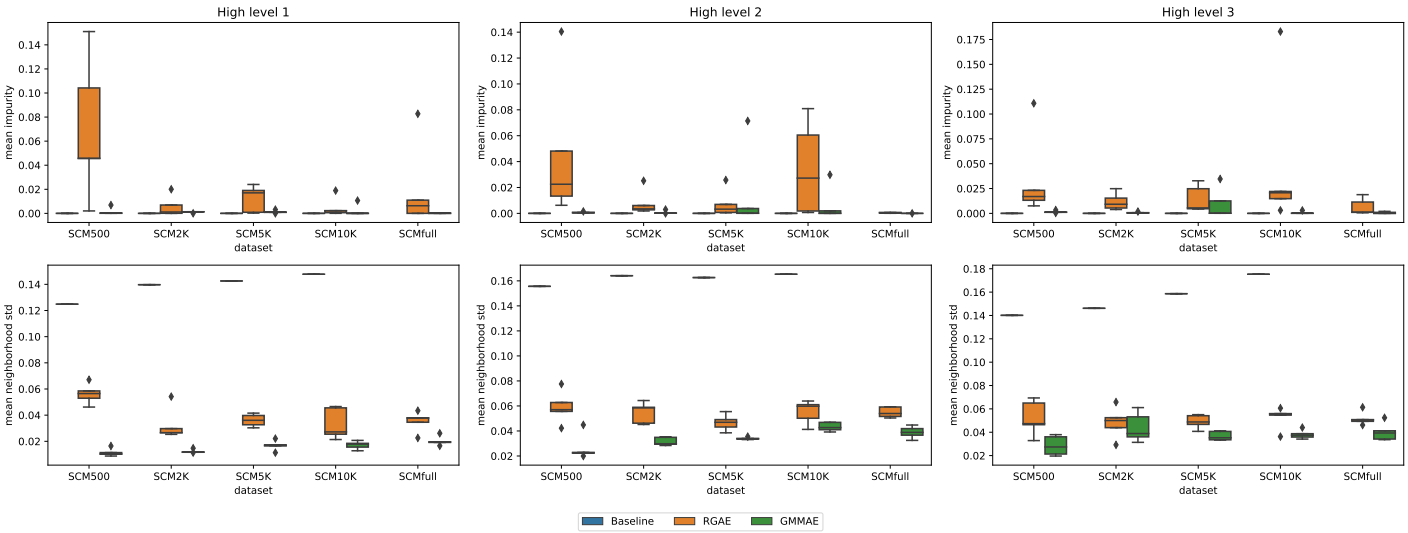

Figure 16: A comparison of the performances of three distinct models, namely the Baseline, RGAE, and GMMAE models, concerning homophily, is presented above. The evaluation is carried out for the first three high levels of abstraction on SCM data sets. The mean impurity (top row) is computed using Equation 19, whereas the mean neighborhood standard deviation (bottom row) is determined using Equation 18. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

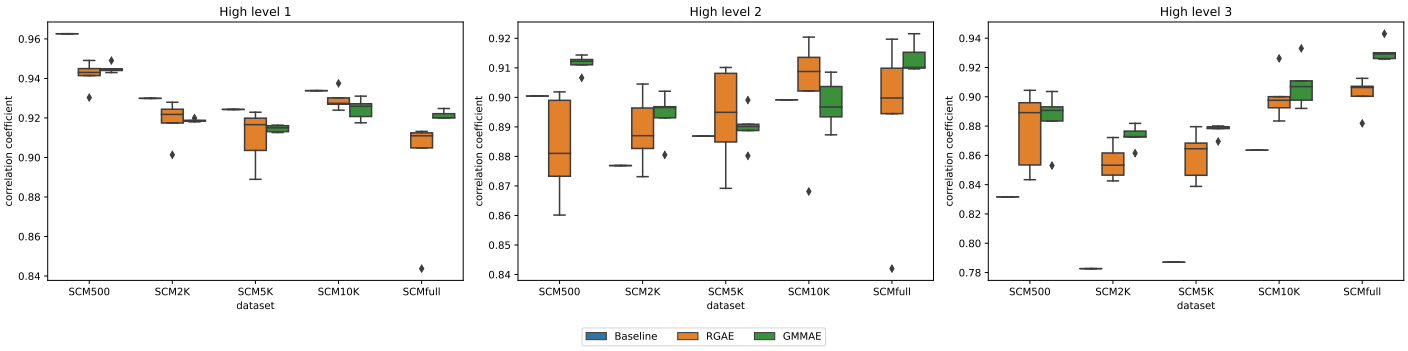

Figure 17: A comparison of the performances of three distinct models, namely the Baseline, RGAE, and GMMAE models, concerning the preservation of the distances, is presented above. The evaluation is carried out for the first three high levels of abstraction on SCM data sets. The mean correlation coefficient is computed using Equation 17. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

GMMAE performances are close but GMMAE outperforms RGAE in most of the experiments. Both are actually doing quite well in this task since they are using RGCN and hence benefit from Theorem 1.

Secondly we can check how well the distances are preserved from one abstracted graph to another with the mean correlation coefficient defined in Equation 17. The results are reported in Figure 17. Overall the correlation coefficients are high, higher than 0.84 for RGAE and GMMAE models, which means that the distances are well preserved between two successive abstracted graphs. While GMMAE model exhibits better results in terms of consistency (high median and small standard deviation), preserving the distances is mostly a task performed by the pooler algorithm with a strong constraint on how the nodes should be merged, this explains why there does not exist a significant difference between the different methods. This is true for high level 1 and 2, while for high level 3 RGAE and GMMAE models outperform the baseline model. One interesting thing, one could notice is that the mean correlation coefficient is decreasing with respect to the level of abstraction (i.e. high level). It would suggest that it becomes more difficult to preserve the distances in the higher levels of abstraction.

Thirdly, the efficiency of the methods is evaluated through the compression factor. To put it another way, how far the methods are able to compress the graph. The results are reported in Figure 18. Regarding the first high level, the GMMAE model outperforms the other models and exhibits a high consistency. In the second level of abstraction GMMAE and RGAE models are still doing well but Baseline model has a higher compression factor. The same observation holds for the third high level, even though the difference between the Baseline and the other models is less important. Those results highlight an interesting property on the structure of SCM, namely that the compression factor is not

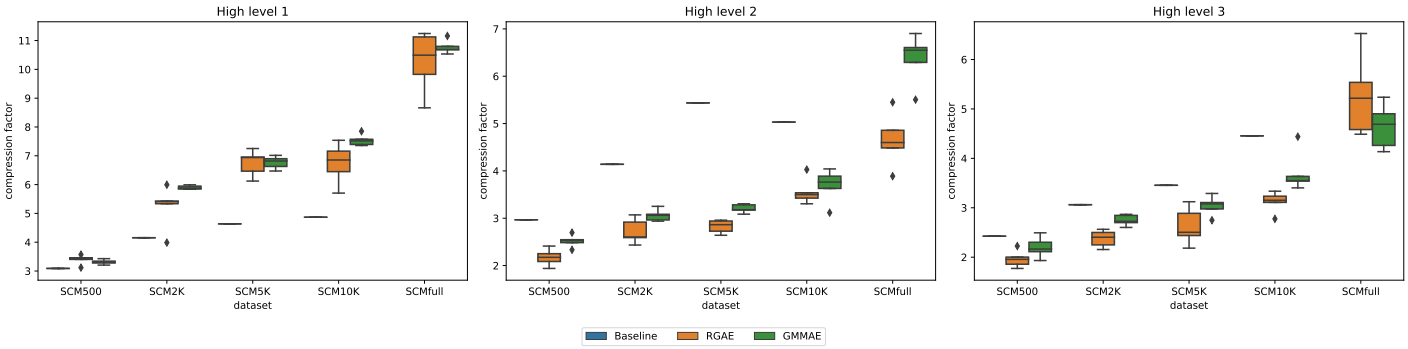

Figure 18: A comparison of the performances of three distinct models, namely the Baseline, RGAE, and GMMAE models, concerning their compression capacity, is presented above. The evaluation is carried out for the first three high levels of abstraction on SCM data sets. The compression factor is computed using Equation 21. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

Dataset	High Level	Model	AUC	AP	Sil-score	Time (s)
SCM500	1	RGAE	0.97 ± 6e-04	0.96 ± 1e-03	0.55 ± 0.07	120.81 ± 134.75
		GMMAE	0.95 ± 2e-03	0.93 ± 2e-03	1.0 ± 5e-04	20.73 ± 0.07
	2	RGAE	0.97 ± 0.01	0.96 ± 0.01	0.47 ± 0.02	10.93 ± 1.01
		GMMAE	0.93 ± 0.01	0.91 ± 0.01	1.0 ± 4e-04	12.72 ± 0.21
	3	RGAE	0.97 ± 0.01	0.96 ± 0.01	0.49 ± 0.07	6.43 ± 0.31
		GMMAE	0.93 ± 4e-03	0.91 ± 0.01	0.99 ± 0.0	7.18 ± 1.29
SCM2K	1	RGAE	0.98 ± 0.01	0.97 ± 0.01	0.6 ± 0.03	32.85 ± 1.2
		GMMAE	0.96 ± 9e-04	0.94 ± 1e-03	0.99 ± 2e-0	32.01 ± 0.19
	2	RGAE	0.98 ± 0.01	0.97 ± 0.01	0.44 ± 0.05	21.83 ± 0.84
		GMMAE	0.94 ± 0.01	0.92 ± 0.01	0.98 ± 0.02	20.75 ± 0.35
	3	RGAE	0.97 ± 5e-03	0.96 ± 0.01	0.47 ± 0.08	15.33 ± 0.75
		GMMAE	0.93 ± 0.01	0.91 ± 0.01	0.98 ± 0.01	13.85 ± 0.55
SCM5K	1	RGAE	0.99 ± 5e-04	0.98 ± 8e-04	0.55 ± 0.03	73.12 ± 3.65
		GMMAE	0.96 ± 2e-03	0.95 ± 2e-03	0.97 ± 0.02	62.65 ± 1.94
	2	RGAE	0.98 ± 4e-03	0.98 ± 0.01	0.48 ± 0.06	32.04 ± 1.14
		GMMAE	0.95 ± 0.01	0.93 ± 0.01	0.97 ± 0.01	30.41 ± 3.12
	3	RGAE	0.98 ± 3e-03	0.97 ± 4e-03	0.46 ± 0.03	24.59 ± 1.51
		GMMAE	0.94 ± 0.01	0.92 ± 0.01	0.96 ± 0.0	24.57 ± 3.47
SCM10K	1	RGAE	0.99 ± 1e-03	0.98 ± 2e-03	0.58 ± 0.0	146.38 ± 10.62
		GMMAE	0.96 ± 2e-03	0.95 ± 3e-03	0.97 ± 0.0	131.12 ± 1.53
	2	RGAE	0.98 ± 3e-03	0.98 ± 3e-03	0.52 ± 0.04	41.77 ± 1.25
		GMMAE	0.96 ± 3e-03	0.94 ± 4e-03	0.96 ± 0.02	41.78 ± 4.47
	3	RGAE	0.98 ± 2e-03	0.97 ± 3e-03	0.54 ± 0.07	32.35 ± 1.89
		GMMAE	0.95 ± 0.01	0.94 ± 0.01	0.98 ± 0.01	30.37 ± 1.26
SCMfull	1	RGAE	0.99 ± 6e-04	0.99 ± 1e-03	0.56 ± 0.04	2620.21 ± 206.78
		GMMAE	0.96 ± 4e-03	0.95 ± 4e-03	0.98 ± 0.01	2496.39 ± 121.13
	2	RGAE	0.99 ± 1e-03	0.99 ± 2e-03	0.5 ± 0.06	604.23 ± 99.84
		GMMAE	0.97 ± 1e-03	0.96 ± 2e-03	0.97 ± 0.01	426.86 ± 91.86
	3	RGAE	0.99 ± 2e-03	0.99 ± 2e-03	0.57 ± 0.09	132.34 ± 17.31
		GMMAE	0.97 ± 3e-03	0.95 ± 4e-03	0.97 ± 0.01	117.54 ± 7.94

Table 3: This table presents the performance analysis of the RGAE and GMMAE models concerning model-related metrics and time performances on SCM data sets. The Area Under the Curve (AUC) is computed using Equation 15, the Average Precision (AP) is determined using Equation 16, and the Silhouette score is calculated using Equation 20. The evaluation is carried out for the first three high levels of abstraction.The experimental results are obtained by performing five repetitions with different seeds on the test set. The VGAE parameters used are provided in Table 2.

constant with respect to the size of the graph. It is actually increasing with respect to the size of the graph. So one could expect that our methods on bigger graphs with the same structure exhibit a higher compression factor.

Finally all the other metrics are reported in Table 3. The baseline model is not reported since most of those metrics are mainly VGAE-related. One can first check if requirement 1 is satisfied by examining the silhouette score. The difference of silhouette scores between the GMMAE and the RGAE is clear, GMMAE always achieves a silhouette score 0.4 higher than what RGAE could achieve. Furthermore, GMMAE is very close to the upper bound of the silhouette score, namely 1, suggesting that the data distribution in the latent space is highly cluster-friendly. As one will notice we will still achieve a high silhouette score for WNS data set, however it will not be as close to 1. This actually gives us some information about SCM data sets, namely that the hypothesis that the nodes can be clustered based on their local structure is very likely. Then regarding the performances of the Auto-Encoder itself, unsurprisingly RGAE outperforms GMMAE, even though the difference is not significant. This is mainly due to the constraints imposed on the latent space of the VGAE. Finally, in terms of time performances, the difference is not huge. The way the VGAE is trained is not that far from the way the RGAE is trained, namely only some EM steps are added to the training. This suggests that RGAE model should be slightly faster than GMMAE model. The results contradict this statement, and this

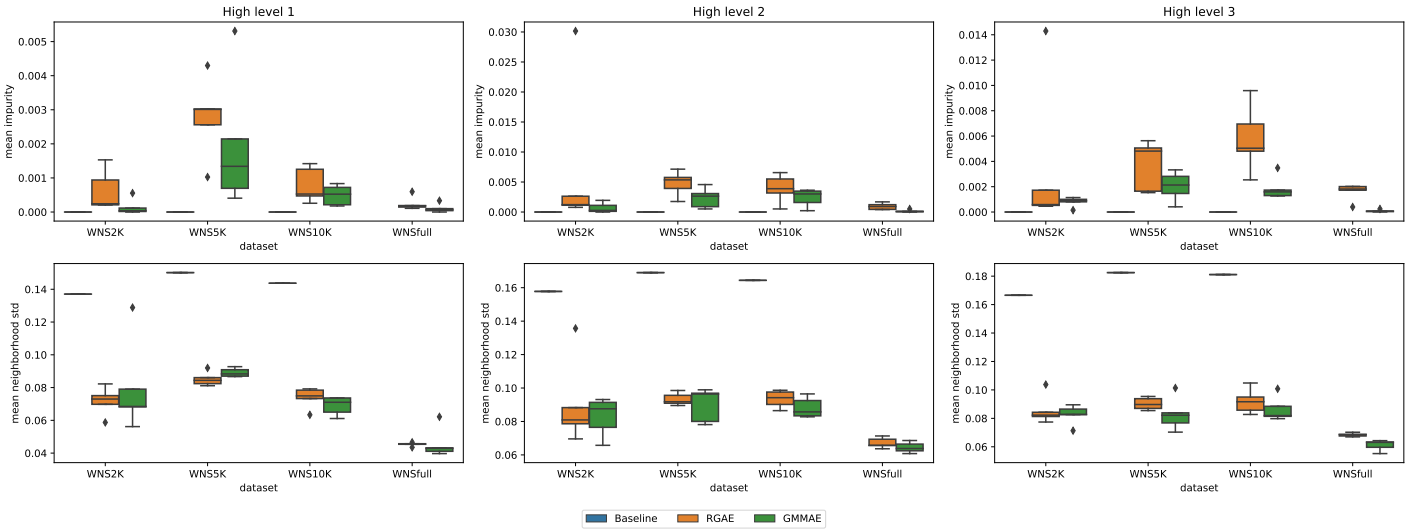

Figure 19: A comparison of the performances of three distinct models, namely the Baseline, RGAE, and GMMAE models, concerning homophily, is presented above. The evaluation is carried out for the first three high levels of abstraction on WNS data sets. The mean impurity (top row) is computed using Equation 19, whereas the mean neighborhood standard deviation (bottom row) is determined using Equation 18. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

could be explained by the fact that the GMMAE model is producing cluster labels that are faster to process for the superpooler algorithm. Thus the time difference does not come from the training part but rather from the pooling part.

Overall GMMAE, RGAE shows far better performances than the baseline model except for the homophily and compression factor, even though the difference between the GMMAE and the baseline model is not significant. Furthermore, GMMAE is the only model satisfying all the requirements stated in section 2 on SCM data sets.

6.3.2 WNS

We repeated exactly the same experiments on WNS data set. First one can inspect the homophily metrics presented on Figure 19. Regarding the mean impurity, more or less the same observations hold, namely that the GMMAE model beats RGAE model. Regarding the mean neighborhood standard deviation, GMMAE and RGAE ,benefiting from the RGCN properties, provide far better results than the Baseline model. The mean neighborhood standard deviation of the GMMAE model is slightly

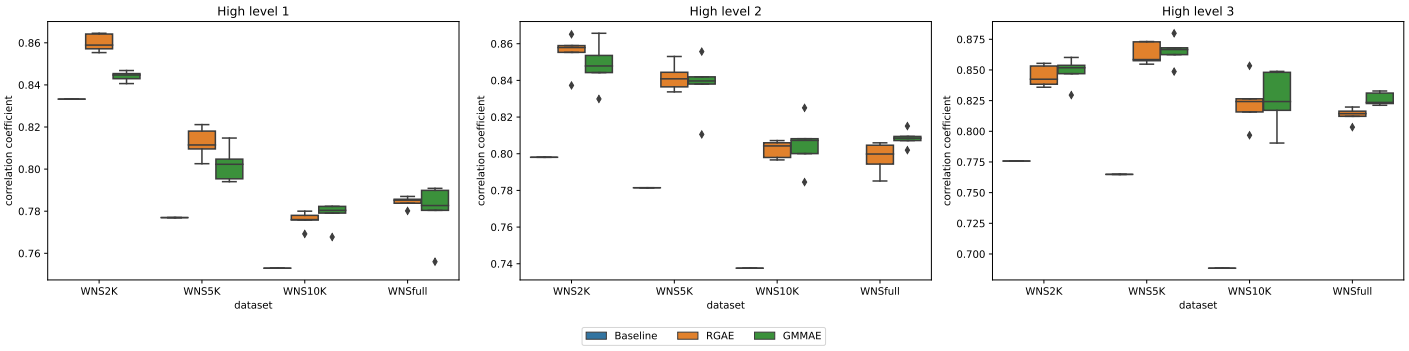

Figure 20: A comparison of the performances of three distinct models, namely the Baseline, RGAE, and GMMAE models, concerning the preservation of the distances, is presented above. The evaluation is carried out for the first three high levels of abstraction on WNS data sets. The mean correlation coefficient is computed using Equation 17. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

higher on WNS than it was on SCM (0.08 against 0.04 on average), but still not important.

Secondly, let's analyze how the distances are preserved in WNS. The results are presented in Figure 20. First the correlation coefficients are lower than the ones obtained in the SCM case but does not decrease (at least for RGAE and GMMAE models) as the level of abstraction increases. While RGAE and GMMAE performances are close, baseline model shows bad results, suggesting that the clustering operation may have its role in preserving the graph distances.

Thirdly, the efficiency of the methods on WNS are presented in Figure 21, through the compression factor. The baseline still outperforms the other models, however recall that such a model is exhibiting very bad performances in term of local structure identification (namely high mean standard deviation) and distance preserving. Thus a model that is showing good results only in terms of compression capacity is not a good model for us. Compression factors of GMMAE and RGAE models are from the same order of magnitude, though GMMAE model exhibits slightly higher ones than RGAE. Overall the compression factors are less important on WNS than the ones observed on SCM (approximately 3 against 10 on the full graph at the first level of abstraction). Besides, even though the compression factor is also increasing with respect to the size of the graph, the slope is lower than the one in the SCM case.

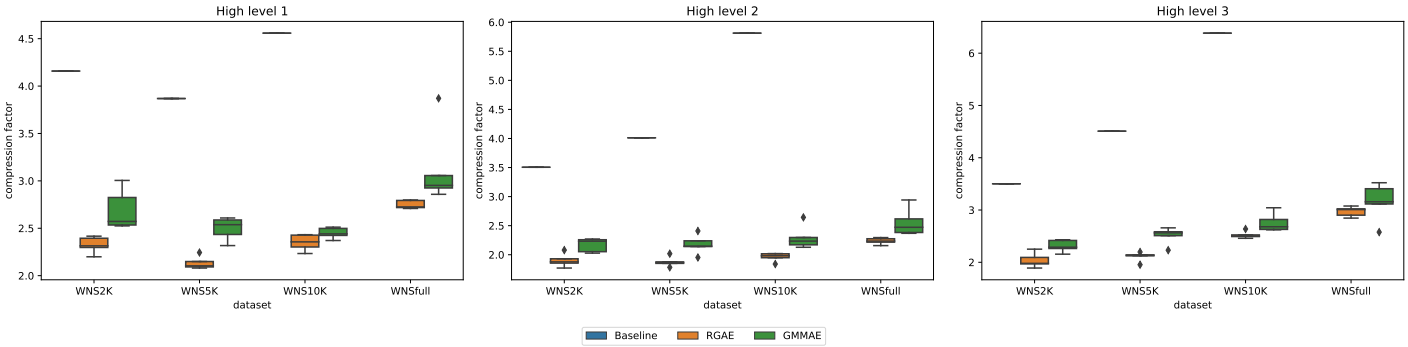

Figure 21: A comparison of the performances of three distinct models, namely the Baseline, RGAE, and GMMAE models, concerning their compression capacity, is presented above. The evaluation is carried out for the first three high levels of abstraction on WNS data sets. The compression factor is computed using Equation 21. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

Dataset	High Level	Model	AUC	AP	Sil-score	Time (s)
WNS2K	1	RGAE	0.96 ± 0.01	0.95 ± 0.01	0.21 ± 0.04	36.47 ± 1.09
		GMMAE	0.77 ± 0.03	0.73 ± 0.02	0.74 ± 0.18	39.33 ± 1.61
	2	RGAE	0.89 ± 0.12	0.87 ± 0.12	0.16 ± 0.0	28.54 ± 0.34
		GMMAE	0.75 ± 0.04	0.72 ± 0.02	0.73 ± 0.23	28.4 ± 0.17
	3	RGAE	0.94 ± 0.01	0.92 ± 0.01	0.17 ± 0.04	24.57 ± 0.33
		GMMAE	0.81 ± 0.04	0.79 ± 0.04	0.7 ± 0.23	22.75 ± 0.68
WNS5K	1	RGAE	0.97 ± 0.01	0.95 ± 0.01	0.15 ± 0.03	67.88 ± 1.59
		GMMAE	0.78 ± 0.02	0.73 ± 0.02	0.78 ± 0.0	75.0 ± 2.06
	2	RGAE	0.95 ± 1e-03	0.93 ± 1e-03	0.15 ± 0.03	40.45 ± 0.95
		GMMAE	0.79 ± 0.04	0.76 ± 0.03	0.78 ± 0.19	39.97 ± 0.83
	3	RGAE	0.93 ± 4e-03	0.91 ± 4e-03	0.13 ± 0.01	35.18 ± 0.45
		GMMAE	0.87 ± 0.01	0.85 ± 0.01	0.84 ± 0.04	34.6 ± 1.74
WNS10K	1	RGAE	0.95 ± 0.01	0.94 ± 0.02	0.18 ± 0.02	109.18 ± 4.33
		GMMAE	0.84 ± 0.01	0.79 ± 0.02	0.48 ± 0.08	120.03 ± 6.14
	2	RGAE	0.93 ± 3e-03	0.91 ± 4e-03	0.13 ± 0.01	65.5 ± 3.3
		GMMAE	0.83 ± 0.02	0.78 ± 0.02	0.64 ± 0.0	64.64 ± 2.25
	3	RGAE	0.92 ± 5e-03	0.9 ± 5e-03	0.12 ± 0.02	53.22 ± 1.99
		GMMAE	0.87 ± 0.02	0.85 ± 0.02	0.69 ± 0.14	56.68 ± 4.81
WNSfull	1	RGAE	0.99 ± 1e-03	0.99 ± 2e-03	0.16 ± 0.01	1024.76 ± 9.3
		GMMAE	0.92 ± 0.02	0.9 ± 0.03	0.78 ± 0.05	1144.8 ± 142.66
	2	RGAE	0.98 ± 1e-03	0.97 ± 2e-03	0.13 ± 0.01	390.97 ± 10.57
		GMMAE	0.89 ± 0.01	0.86 ± 0.01	0.79 ± 0.07	443.31 ± 136.26
	3	RGAE	0.97 ± 3e-03	0.96 ± 3e-03	0.15 ± 0.02	1057.33 ± 670.21
		GMMAE	0.92 ± 0.01	0.91 ± 0.01	0.84 ± 0.05	669.75 ± 422.93

Table 4: This table presents the performance analysis of the RGAE and GMMAE models concerning model-related metrics and time performances on WNS data sets. The Area Under the Curve (AUC) is computed using Equation 15, the Average Precision (AP) is determined using Equation 16, and the Silhouette score is calculated using Equation 20. The evaluation is carried out for the first three high levels of abstraction.The experimental results are obtained by performing five repetitions with different seeds on the test set. The VGAE parameters used are provided in Table 2.

Finally all the other metrics are reported in Table 4. The silhouette score of the clustering provided by GMMAE model is far higher than the one provided by the RGAE model (a least 0.3 higher). However this time it is not as close to 1 as it was for the SCM, suggesting that the hypothesis that can be clustered based on their local structure is less likely. As expected the Auto-Encoder performances of the RGAE model are better than the GMMAE model. The gap between the two models is higher on WNS data set, which confirms the hypothesis that modeling the latent space with a multi-modal distribution is not the best model for this data set. In terms of time performances, The RGAE model is this time slightly faster than GMMAE one, even though performances are close.

6.4 Methods robustness

It remains one metric presented in subsection 5.4 that has not been inspected, namely the robustness of the methods. The baseline model is not affected by this metric since it is a deterministic model and then will always return the same output. The robustness of the GMMAE and RGAE models are assessed on both data sets using the V-measure.

The results on SCM are presented in Figure 22. GMMAE model outperforms RGAE model in terms of V-measure. This could have been inferred from the previous results on SCM, where GMMAE was exhibiting lower variances than RGAE. For high level 1, GMMAE almost reaches the V-measure upper bound for both clusterings. As one may notice the V-measure decreases with respect to the level of abstraction, especially for the pooling V-measure. This is mainly due to the propagation of the difference of clustering. That is to say: all the repetitions are initially starting with the same graph and generate

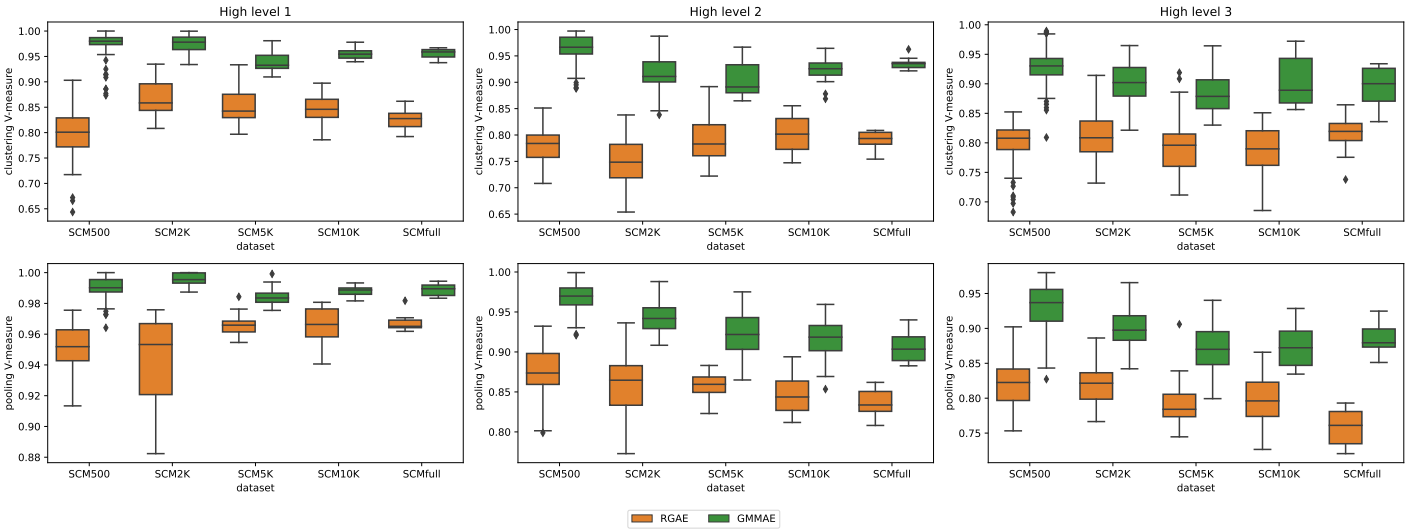

Figure 22: A comparison of the performances of RGAE, and GMMAE models, concerning their robustness, is presented above. The evaluation is carried out for the first three high levels of abstraction on SCM data sets. The clustering V-measure (top row) and pooling V-measure (bottom row) are computed as described in subsubsection 5.4.6. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

first abstracted graphs. Those abstracted graphs may be slightly different and the same methods are reapplied on them to get an abstracted graph of level 2. The difference observed at the first level of abstraction may then be amplified in the second level and lead to abstracted graph of level 2 even more different. This way, one can explain why V-measure is decreasing from one high level to another.

The results on WNS are reported in Figure 23. The V-measure are lower and consistency of GMMAE is only slightly better then RGAE if not similar to it. A lower V-measure could have been expected since the results showed that modelling the distribution of the latent space with a multimodal one was harder on WNS than on SCM. Regarding the evolution of the V-measure across the levels of abstraction, the same trend is observed, namely that V-measure is descreasing. In WNS case, the slope is even higher.

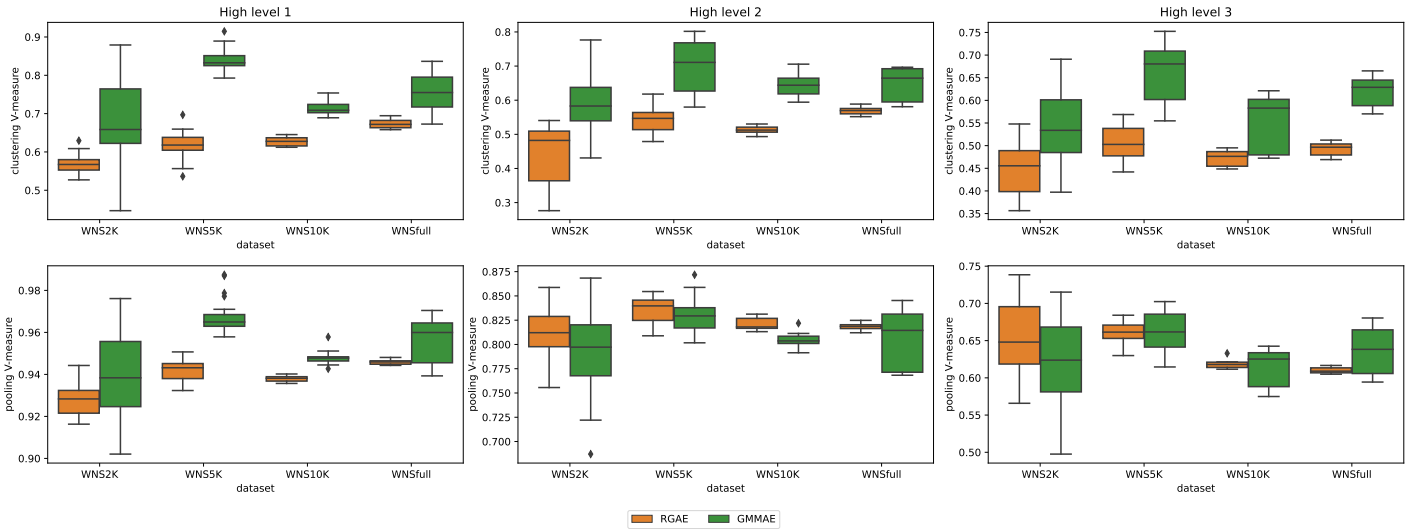

Figure 23: A comparison of the performances of RGAE, and GMMAE models, concerning their robustness, is presented above. The evaluation is carried out for the first three high levels of abstraction on WNS data sets. The clustering V-measure (top row) and pooling V-measure (bottom row) are computed as described in subsubsection 5.4.6. The experimental results are obtained by conducting five repetitions with different seeds on the test set. The VGAE parameters used here are provided in Table 2.

Finally overall GMMAE exhibits higher robustness than a classical GAE such as RGAE. Furthermore, it has a very high robustness on SCM, very close to 1, which is actually a very good property since it would be very undesirable to have far different results from one run to another.

6.5 Scaling-up potentialities

The last point to be discussed is how far the methods can be applied to larger graph using method described in subsection 4.4. Recall that the goal is to split the whole graph into smaller subgraphs, such that applying the VGAE to each subgraph separately does not impact the embeddings computed by the model. To do so METIS algorithm is used to partition the graph into subgraphs, and then higher subgraphs are generated from those latter such that it includes the k-hop neighborhoods of specific nodes, namely the induced subgraphs. We are actually interested in how far we can reduce the size of those induced subgraphs.

The experiments have been performed on SCM, called SCM1 in this section, and on a larger graph also coming from SwissCom data base, namely SCM2. They will also be performed on the union of SCM1 and SCM2, namely SCM[1 + 2]. We recorded the size of the largest induced subgraph as well as the mean size for different number of groups (input of METIS algorithm). They were performed for different values of k. The results are presented in Figure 24.

Regarding SCM1, the size of the largest induced subgraph for $k = 2$ stailizes at approximately 40% of the size of the whole graph, meaning that one can divide by 2.5 two the size of the original graph. On SCM2 and SCM3, the size of the largest induced subgraph stabilizes at a lower value close to 30%. Though one could have expected better results, it is still satisfying since if one divides by n the size of the input graph, the required gpu memory is also divided by n (assuming a linear relationship between the number of the nodes and the number of edges, which is approximately the case for SCM data set).

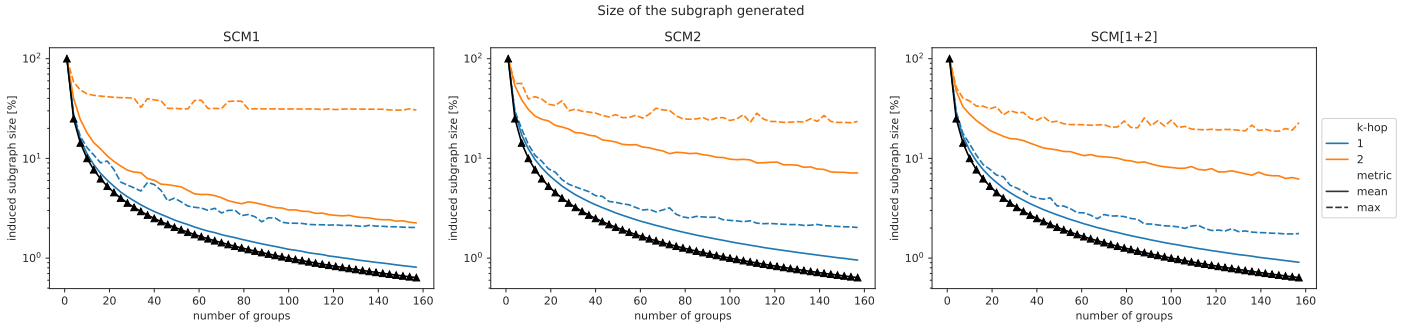

Figure 24: The above plots are showing the size of the induced graphs generated using algorithm 6. The experiments were conducted on SwissCom data sets SCM1, SCM2 and SCM[1+2] for $k = 1$ and 2. The black curve represents the inverse function (i.e. $1/x$) and represents more or less the number of useful nodes in the subgraph generated.

7 Discussion

This work provided a good baseline, namely GMMAE model, to generate a sequence of abstracted graphs. While satisfying the requirements stated in section 2, and showing quite good results on SCM, those methods are still suffering from some undesirable effects. We empirically noticed, on graph of small sizes, that setting a very high value for the initial number of clusters (¿ 100) may cause the explosion of the KL loss and then force to restart the training. Furthermore we were not facing issues with the node pooler algorithm when applied to the clustering assignment output by GMMAE model, but issues were experienced when feeding the node pooler algorithm with the clustering assignment output by RGAE model. The issue was related to the NP-hard aspect of the maximal cliques algorithm included in the node pooler algorithm. Thus while exhibiting good performances, those methods could be greatly improved in many ways.

7.1 Further improvements

The first straightforward improvement to perform is to fine-tune the hyperparameters of the method and more specifically the weights in the loss. Indeed the hyperparameters were chosen based on the intuition more than using rigorous methods. Furthermore using the same hyperparameters from one level of abstraction to another, may not be optimal since the structure and the size of the graph is modified. Hence, finding a procedure that automatically re-adapts the hyperparameters depending on the characteristics of the abstracted graphs generated would be awesome.

Secondly we propose a new model to improve the performances of the VGAE. It is still a VGAE but first it does make the assumption stated in Equation 10, namely that $Z|A, X$ and $c|A, X$ are independent, and secondly it does not approximate π_i^* as done in Equation 11. Indeed if the intuition introduced in [27] suggests to compute π_i^* to minimize the KL loss, we think that this is actually not the right way to proceed since minimizing only the KL will not allow us to reach the objective of the VGAE. Also we think that considering that $Z|A, X$ does not depend on $c|A, X$ is an important assumption that may hamper the performances of the VGAE. Thus in our new model the inference model can be factorized as follows using baye's rule:

$$q(Z, c|A, X) = q(Z|c, A, X)q(c|A, X)$$

In other words, in the new model, we will first sample c_i according to π^* for each node, and depending on c_i determining μ_i, σ_i and finally sample z_i. Such a process is described in Figure 25. The generative process of the new VGAE is unchanged and stick to the one described in definition 12, while the inference process is slightly modified as follows:

$$q(Z|c, A, X) = \prod_{i=1}^{N} q(z_i|c_i, A, X) \quad \text{where} \quad q(z_i|c_i, A, X) = \mathcal{N}\big(z_i; \mu_i, \text{diag}(\sigma_i^2)\big)$$

$$q(c|A, X) = \prod_{i=1}^{N} q(c_i|A, X) \quad \text{where} \quad q(c_i|A, X) = \text{Cat}(\pi_i^*)$$

$$\text{with} \quad \Pi^* = f_\pi(A, X) \quad \text{and} \quad \mu, \log(\sigma) = f_{\mu,\sigma}(A, X, \Pi^*)$$

Thus now μ and σ are a function of Π^*. Actually, it will be more accurate to say that μ and σ are a function of c, where c is a function of Π^*. We come back to the tricky point where one should sample c_i using a categorical distribution parameterized by π_i^* without breaking the operator graph. The reparametrization trick was working well with continuous distributions, however it's harder with discrete ones. Let's first state what one is looking for: we are searching for a random variable X generated independently of π^* and a transformation h that takes as input X and π^* and returns a random variable Y that satisfies:

$$Y = h(X, \pi^*) \quad Y \sim \text{Cat}(\pi^*)$$

We will first replace Y by a vector of length C, namely $\bar{Y} = [\bar{Y}_1, ...\bar{Y}_C]$. This vector can only take one hot vector values such that one can infer Y from $\bar{Y}$. Then we will sample X from a continuous

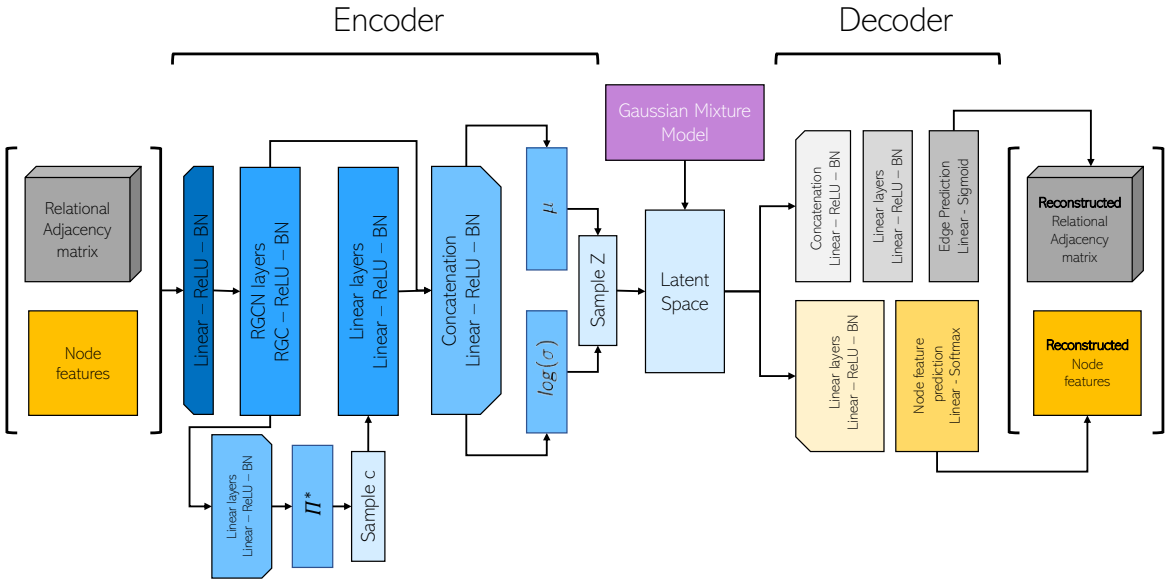

Figure 25: The upgraded version of the VGAE architecture is presented above. The decoder is similar to the one presented in Figure 5. The encoder structure is revisited such that μ and $\log(\sigma)$ are now a function of c. Thus c is first sampled as a one-hot vector, concatenated to another vector and a MLP-like layer is used to compute μ and $\log(\sigma)$.

uniform distribution between [0,1] and apply the following transformation:

$$X \sim \mathcal{U}(0,1) \quad \bar{Y}_i = \mathbb{1}_{X \in [\alpha_{i-1}, \alpha_i[} \quad \text{with} \quad \alpha_i = \sum_{j=1}^{i} \pi^*_{.,j}$$

$$\mathbb{P}(Y = c) = \mathbb{P}(\bar{Y}_c = 1) = \pi^*_{.,c}$$

Unfortunately the indicator function is not differentiable, thus we will approximate it with the sigmoid function such that:

$$\bar{Y}_i = \mathbb{1}_{X \in [\alpha_{i-1}, \alpha_i[} = \mathbb{1}_{X \geq \alpha_{i-1}} - \mathbb{1}_{X \geq \alpha_i} \approx \sigma_\beta(X - \alpha_{i-1}) - \sigma_\beta(X - \alpha_i)$$

$$\text{with} \quad \sigma_\beta(x) = \frac{1}{1 - e^{-\beta x}}$$

The higher β, the better the approximation in terms of absolute difference. However a higher β also means a lower gradient with respect to π^*. Thus this parameter should be set carefully to achieve a good approximation while avoiding gradient vanishing. One may wonder why approximated the one-hot vector $\bar{Y}$ by sigmoid function will not impact that much the output of the VGAE ? To justify this, let's assume batchnorm is not used in architecture presented in Figure 25, then let's denote $\lambda_{max,i}$ the maximum singular value of layer k weight matrix. Assuming ReLU activation function is used, $f_{\mu,\sigma}$ satisfies:

$$\| f_{\mu,\sigma}(\bar{Y}_1) - f_{\mu,\sigma}(\bar{Y}_2) \| \leq \left[\prod_i \lambda_{max,i} \right] \| \bar{Y}_1 - \bar{Y}_2 \|$$

Thus if the approximation of $\bar{Y}$ is good enough and the singular values are reasonably not large (which could be controlled using spectral normalization), then the output of the VGAE will be well approximated as well. Regarding the training part of this new VGAE we recommend to delay the initialization as it was done previously. Once the GMM is initialized, we recommend to perform a first pre-training step of the linear layers predicting Π^*, such that Π^* matches Π and thus avoiding an explosion of the KL loss at the beginning. We finally recommend to start without reassigning cluster during the training as it will require Π^* to re-adapt to the new clusters. Finally regarding the modules used in the architecture of the VGAE, they could be improved using other graph specific operations such as Graph attention networks [14] that showed better performances than GCNs (in our context we will rather use RGAT to replace RGCN).

The last improvement to be performed, but not least, is to review the pooler algorithm. Indeed the NP-hard algorithm that is searching for maximal cliques, may cause disasters in terms of time performances depending on the structure of graph. A great improvement will be to find another deterministic algorithm that is still satisfying constraint described in subsection 4.2.

7.2 Future work

This first step would be to implement and test the modifications mentioned previously. Then it would be necessary to develop, on the top of the multi-abstraction module, a streaming module. A streaming module is a module that will be able to compute efficiently the different levels of abstraction for a graph that evolves in real time. Namely given a time series of graph $\{\mathcal{G}^{(t_1)}...\mathcal{G}^{(t_P)}\}$, how can we efficiently compute the abstracted graph time series of the first level of abstraction $\{\mathcal{G}_{(1)}^{(t_1)}...\mathcal{G}_{(1)}^{(t_P)}\}$?

Under the assumption that the difference between two successive graphs is small enough (only a few nodes and edges added/removed), one can actually leverage the RGCN property presented in Theorem 3. In other words if a node is added or removed, only the embeddings of the nodes that are in the neighborhood of this node will be impacted. Thus we recommend to sequentially build the abstracted graph time series, such that the multi-level abstraction is applied once to build the abstracted graph of $\mathcal{G}^{(t_1)}$ and the computations of the other ones are done based on this latter. On the top of this suggestion one should also imagine how one can go to higher levels of abstraction.

Once the streaming module would have been developed, we think that the whole module would be ready to be deployed. We do not hide that we strongly recommend to re-implement the methods presented in this work. Indeed, even though the code works well, it is not optimized and was developed for prototyping purposes. Many parts in the code may be useless and clutter the modules. To speed-up the methods, we would recommend to re-implement the pooling module in C++ since it requires a few libraries in python that also exist in C++.

8 Conclusion

In this work, the objective was to build a sequence of increasingly smaller KGs starting from an original large KG, performing only grouping operations between the nodes. Three properties were required for this sequence: the nodes should be grouped based on an unsupervised method nodes that are grouped together should exhibit a similar neighborhood, and the graph distances between two successive KGs in the sequence should be relatively preserved. The first two requirements were satisfied using an unsupervised GNN, namely a VGAE, while a custom algorithm was developed to reach the third one. The results confirmed that the three requirements are fulfilled by the methods presented in this work. Furthermore they showed good performances on SCM data sets compared to other more basic methods. The results were slightly lower on WNS data sets but still better than other methods.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[2] M. E. J. Newman, "Spectral methods for community detection and graph partitioning," *Physical Review E*, vol. 88, no. 4, 2013.

[3] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "VOG: summarizing and understanding large graphs," in *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014* (M. J. Zaki, Z. Obradovic, P. Tan, A. Banerjee, C. Kamath, and S. Parthasarathy, eds.), pp. 91–99, SIAM, 2014.

[4] Y. Lim, U. Kang, and C. Faloutsos, "Slashburn: Graph compression and mining beyond caveman communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3077–3089, 2014.

[5] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016* (B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, eds.), pp. 855–864, ACM, 2016.

[6] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014.

[7] U. von Luxburg, "A tutorial on spectral clustering," 2007.

[8] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States* (C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2787–2795, 2013.

[9] M. Nickel, V. Tresp, and H. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011* (L. Getoor and T. Scheffer, eds.), pp. 809–816, Omnipress, 2011.

[10] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[11] X. Glorot, A. Bordes, J. Weston, and Y. Bengio, "A semantic matching energy function for learning with multi-relational data," 2013.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[13] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," 2017.

[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[15] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016.

[16] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang, "Attributed graph clustering: A deep attentional embedding approach," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019* (S. Kraus, ed.), pp. 3670–3676, ijcai.org, 2019.

[17] B. Hui, P. Zhu, and Q. Hu, "Collaborative graph convolutional networks: Unsupervised learning meets semi-supervised learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4215–4222, 04 2020.

[18] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden* (J. Lang, ed.), pp. 2609–2615, ijcai.org, 2018.

[19] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*, ACM, 2019.

[20] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[21] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 1024–1034, 2017.

[24] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational graph attention networks," 2019.

[25] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[26] C. Doersch, "Tutorial on variational autoencoders," 2016.

[27] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: An unsupervised and generative approach to clustering," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017* (C. Sierra, ed.), pp. 1965–1972, ijcai.org, 2017.

[28] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[29] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, p. 575–577, 1973.

[30] G. Karypis and V. Kumar, "Metis—a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing ordering of sparse matrices," 1997.

[31] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, (Prague, Czech Republic), pp. 410–420, Association for Computational Linguistics, 2007.

[32] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[33] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[34] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," 2020.

[35] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "GEMSEC: graph embedding with self clustering," in *ASONAM '19: International Conference on Advances in Social Networks Analysis and Mining, Vancouver, British Columbia, Canada, 27-30 August, 2019* (F. Spezzano, W. Chen, and X. Xiao, eds.), pp. 65–72, ACM, 2019.

[36] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," 2020.

[37] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (S. A. McIlraith and K. Q. Weinberger, eds.), pp. 1811–1818, AAAI Press, 2018.

9 Appendix

9.1 Computation of the KL loss

For the sake of simplicity we will omit A, X in the inference model and replace $q(Z, c|A, X)$ by $q(Z, c)$.

$$\begin{aligned}
D_{KL}[q(Z,c)||p(Z,c)] &= \int_{Z,c} q(Z,c)\log\left(\frac{q(Z,c)}{p(Z,c)}\right)dZdc \\
&= \int_{Z,c}\prod_{j=1}^{N} q(z_j,c_j)\sum_{i=1}^{N}\log\left(\frac{q(z_i,c_i)}{p(z_i,c_i)}\right)dZdc \\
&= \sum_{i=1}^{N}\int_{z_i,c_i} q(z_i,c_i)\log\left(\frac{q(z_i,c_i)}{p(z_i,c_i)}\right)\left[\prod_{j\neq i}\int_{c_j,z_j} q(z_j,c_j)dz_jdc_j\right]dz_idc_i \qquad (22)\\
&= \sum_{i=1}^{N}\int_{z_i,c_i} q(z_i,c_i)\log\left(\frac{q(z_i,c_i)}{p(z_i,c_i)}\right)dz_idc_i \\
&= \sum_{i=1}^{N} D_{KL}[q(z_i,c_i)||p(z_i,c_i)]
\end{aligned}$$

With notations $q(c) = \pi_c^*$ and $p(c) = \pi_c$, let's now compute $D_{KL}[q(z,c)||p(z,c)]$:

$$\begin{aligned}
D_{KL}[q(z,c)||p(z,c)] &= \sum_{c=1}^{C}\int_z q(z,c)\log\left(\frac{q(z,c)}{p(z,c)}\right)dz \\
&= \sum_{c=1}^{C} q(c)\int_z q(z|c)\log\left(\frac{q(z|c)q(c)}{p(z|c)p(c)}\right)dz \\
&= \sum_{c=1}^{C}\pi_c^*\log\left(\frac{\pi_c^*}{\pi_c}\right) + \pi_c^*\int_z q(z|c)\log\left(\frac{q(z|c)}{p(z|c)}\right)dz \qquad (23)\\
&= D_{KL}[\pi^*||\pi] + \sum_{c=1}^{C}\pi_c^* D_{KL}[q(z|c)||p(z|c)] \\
&= D_{KL}[\pi^*||\pi] + \sum_{c=1}^{C}\pi_c^* D_{KL}[\mathcal{N}(\mu, diag(\sigma^2))||\mathcal{N}(\mu_c, diag(\sigma_c^2))]
\end{aligned}$$

Putting Equation 22 and Equation 23 together we get the final result:

$$D_{KL}[q(Z,c|A,X)||p(Z,c)] = \sum_{i=1}^{N}\left[D_{KL}[\pi_i^*||\pi] + \sum_{c=1}^{C}\pi_{i,c}^* D_{KL}[\mathcal{N}(\mu_i, \mathrm{diag}(\sigma_i^2))||\mathcal{N}(\mu_c, \mathrm{diag}(\sigma_c^2))]\right]$$