- ORIGINAL ARTICLE -

Graph Representations for Reinforcement Learning

Representaciones de Grafos para Aprendizaje por Refuerzo

Esteban Schab[1], Carlos Casanova[1,3], and Fabiana Piccoli[2,3]

[1] *Universidad Tecnológica Nacional, Facultad Regional Concepción del Uruguay*
{schabe, casanovac}@frcu.utn.edu.ar
[2] *Universidad Nacional de San Luis, San Luis, Argentina*
mpiccoli@unsl.edu.ar
[3] *Universidad Autónoma de Entre Ríos, Concepción del Uruguay, Argentina*

Abstract

Graph analysis is becoming increasingly important due to the expressive power of graph models and the efficient algorithms available for processing them. Reinforcement Learning is one domain that could benefit from advancements in graph analysis, given that a learning agent may be integrated into an environment that can be represented as a graph. Nevertheless, the structural irregularity of graphs and the lack of prior labels make it difficult to integrate such a model into modern Reinforcement Learning frameworks that rely on artificial neural networks. Graph embedding enables the learning of low-dimensional vector representations that are more suited for machine learning algorithms, while retaining essential graph features. This paper presents a framework for evaluating graph embedding algorithms and their ability to preserve the structure and relevant features of graphs by means of an internal validation metric, without resorting to subsequent tasks that require labels for training. Based on this framework, three defined algorithms that meet the necessary requirements for solving a specific problem of Reinforcement Learning in graphs are selected, analyzed, and compared. These algorithms are Graph2Vec, GL2Vec, and Wavelet Characteristics, with the latter two demonstrating superior performance.

Keywords: Computational Intelligence, Reinforcement Learning, Graph Embeddings, unsupervised GRL, Whole Graph Embedding.

Resumen

El análisis de grafos es un tópico emergente debido a la expresividad de los modelos basados en grafos y al desarrollo de algoritmos para su procesamiento. Un área que puede beneficiarse de estos avances es el aprendizaje por refuerzo, dado que un agente de aprendizaje puede estar inmerso en un entorno modelable como un grafo. Sin embargo, tanto la irregularidad de las características estructurales de los grafos como la ausencia de etiquetas a priori dificultan la incorporación de un modelo de este tipo en los marcos modernos de Aprendizaje por Refuerzo basados en redes neuronales artificiales. En este sentido, los embeddings de grafos permiten aprender representaciones vectoriales de baja dimensión, más adecuadas para los algoritmos de aprendizaje automático, preservando al mismo tiempo las características clave de los grafos. Proponemos un marco para evaluar algoritmos de Graph Embedding y su capacidad para preservar la estructura y características relevantes de los grafos mediante una métrica de validación interna, sin recurrir a tareas posteriores que requieran etiquetas para el entrenamiento. Aplicando este marco con un problema concreto, se seleccionan, analizan y comparan tres algoritmos que cumplen los requisitos necesarios: Graph2Vec, GL2Vec y Wavelet Characteristics, donde los dos últimos muestran un mejor comportamiento.

Palabras claves: Inteligencia Computacional, Aprendizaje por Refuerzo, Embeddings de grafos, GRL no supervisado, Embedding de grafo entero.

1 Introduction

Generally, complex or critical real-world systems lack determinism or dichotomy. Achieving a good computational solution and reasonable response times often requires significant computing power. There may be several factors that contribute to the complexity of a system. For example, real-world replications may have real-world consequences, or the amount of data that needs to be processed may be large and incomprehensible to humans. Furthermore, complex systems may involve interactions between its parts, forming networks or graphs.

Graph analysis has attracted significant attention nowadays as networks proliferate in the real world. Graphs are used to represent information in various domains, like social sciences: friendship and social networks [1], linguistics: word co-occurrence networks [2], electronic commerce, reference networks of academic papers, biology: molecular and protein interaction networks [3], transportation: logistics and vehicular traffic, among others.

Understanding network systems is facilitated by modelling interactions in graphs [4]. Graph analysis has received considerable attention in recent decades because it enables us to understand and use the hidden information they contain. According to [5], there are four categories that summarize classical graph analysis tasks:

1. **Node classification**: The objective is to label each node in the graph based on either the labeled nodes or the network structure [6]. Two approaches have been proposed: classification methods that use random walks for label propagation, and methods that extract features from nodes and use classifiers for labeling.

2. **Link prediction**: This enables us to predict missed or potential links in the future [7]. Different approaches can be employed, we found similarity-based methods, maximum likelihood models, and probabilistic models.

3. **Clustering**: This task is to identify similar entities within the network and group them together [8]. Clustering methods comprise attribute-based models as well as those that maximise distances between clusters.

4. **Visualization**: This helps to understand the structure of the network and identify any changes over time [9].

Efficient network analysis requires finding a concise and effective representation of the data. A graph G is defined as $\langle V, E\rangle$, where V is the set of graph vertices / nodes, and E is the set of edges, each of which represents the relationship between two nodes [10, 5]. Generally, G is represented by a matrix, the adjacency matrix. Each element of the matrix indicates whether the nodes are adjacent or not. A binary matrix represents unweighted edges, i.e. graphs whose edges do not have weight. If each edge has weight, the matrix is non-binary.

For large networks, the conventional graph representation can lead to a bottleneck. Some of the typical problems in this scenario include [11]:

1. High Computational Complexity: in general, the adjacency matrix utilised to model relationships between nodes tends to be high-dimensional and sparsely populated [5]. Thus, most algorithms for analysing or processing networks carry high computational complexity.

2. Low parallelism: Representing network data in a traditional manner presents serious difficulties for the design and implementation of parallel/distributed algorithms. The bottleneck arises from the coupling of nodes in a network, as explicitly shown in E. Distributing these nodes to different processes leads to high communication costs and slows down the speedup ratio. Limited progress has been made in the graph parallelization via subtle segmentation of large-scale graphs [12]. However, the success of these methods heavily relies on the underlying graph's topological characteristics.

3. Machine Learning limitations: Machine Learning methods typically assume that data samples are represented by independent vectors, while network data samples (i.e. nodes) are dependent on each other through E. Additionally, graphs can have variable dimensions and structures, which can be a drawback when analysing a set of graphs.

Aside from the drawbacks already pointed out by [11], the adjacency matrix representation for graphs may contain redundant or noisy information and may have varying dimensions if the problem comprises a set of graphs or a graph that alters over time. A growing approach to solve these issues is Graph Representation Learning (GRL) methods [10]. These methods aim to design or learn low-dimensional vector representations that enable encoding graph information using structures that preserve it, including topology and node features. These representations are commonly referred to as Graph Embeddings. The basic idea is to learn the dense and continuous representations of nodes in a low-dimensional space. This approach aims to diminish noise and redundancy, preserving the intrinsic structure and essential properties of graphs.

Many network analysis problems that are iterative or combinatorial can be addressed in new vector representation spaces by computing mapping functions, distance metrics or operations on the embedding vectors. This results in a reduction of the complexity problem. The elimination of coupling among the nodes allows the possibility of developing good parallel computing solutions for large-scale network analysis. Graph embedding also allows for the application of machine learning to network analysis.

Graph analysis can be a valuable subsidiary task for reinforcement learning (RL), as has been demonstrated in recent years by research such as [13, 14]. Modelling and adjusting the state/observation of the agent is one of the key components of RL algorithms and models. When tackling RL problems in which the agent is present in a networked environment, incorporating a graph that details the network and its relevant characteristics and properties can be a highly promising approach. Given the unique characteristics and requirements of this approach, it can be seen as a fresh graph analysis task distinct from those previously recognised. In this particular scenario, a 'whole graph' representation is required, which allows the dimension to be reduced and standardised for use as input in neural networks and other computational intelligence algorithms composing RL agents [13, 14, 15].

This paper describes the new task mentioned above and its specific requirements for Graph Embedding. We propose a framework to evaluate and compare graph embedding algorithms and their ability to preserve structure and relevant features. Following this framework, and for a specific RL problem in graphs, three state-of-the-art algorithms are selected, analysed and compared. All of them fulfil the necessary requirements described. These are Graph2Vec[16], GL2Vec[17] and Wavelets Characteristic[18].

The paper is organised as follows. The next section provides the basic definitions and notations used. Section 3 introduces GRL, its characteristics and advantages. Section 4 introduces RL as a new class of graph analysis tasks that require GRL as a sub-task. Section 5 proposes a framework for evaluating graph embedding algorithms. Section 6 presents the analysis, and Section 7 presents the results. Finally, Section 8 presents the conclusion and future work. Due to space limitations, there is no specific section on related work; these are referenced throughout the paper.

2 Preliminaries and Definitions

Since the definitions and notation used in graphs usually vary among authors, in this section we introduce those used in this work.

Definition 1 (Graph). A graph G is a 3-tuple (V,E,φ), where V is a set of vertices/nodes, E is a set of edges/links, and φ is a function $\varphi : e_k \rightarrow (v_i, v_j)$ that assigns a pair of vertices $v_i, v_j \in V$ to each edge $e_k \in E$. It is said that when $\varphi(e_k) = (v_i, v_j)$, e_k is positively incident on v_i and negatively incident on v_j. Also, given two arbitrary nodes $v_i, v_j \in V$, v_i is adjacent to v_j if and only if $\exists e_k \in E, \varphi(e_k) = (v_i, v_j)$.

A graph is called undirected if and only if $\forall v_i, v_j \in V$, v_i is adjacent to $v_j \rightarrow v_j$ is adjacent to v_i, i.e., when adjacency relation is symmetric. Else, it is considered directed.

Remark. Some authors consider the existence of *parallel edges* in graphs. Given a graph $G = (V,E,\varphi)$, two edges $e_i, e_j \in E$ are said to be parallel if and only if $\varphi(e_i) = \varphi(e_j)$, i.e., both edges are incident on the same vertices and in the same way.

In addition, a *loop* is an edge $e_k \in E$ such that $\varphi(e_k) = (v_i, v_i)$ for a vertex $v_i \in V$, i.e., an edge that is positively and negatively incident on the same vertex.

A graph without parallel edges and loops is called a *simple graph*. From now on, in this work, graph is understood as simple graph.

Definition 2 (Adjacency Matrix). A finite graph $G = (V,E,\varphi)$ can be represented as a $|V| \times |V|$ adjacency matrix A, where each $a_{ij} = 1$ if v_i is adjacent to v_j, and 0 otherwise.

Definition 3 (Weighted graph). A weighted graph is a 4-tuple (V,E,φ,wf), where (V,E,φ) is a graph and wf is a weight function $wf : e_i \rightarrow w_i$ that assigns a weight $w_i \in \mathbb{R}$ to each edge $e_i \in E$.

A finite weighted graph can be represented as a $|V| \times |V|$ weight matrix W, assigning $w_{i,j} = wf(e_k)$ if $\varphi(e_k) = (v_i, v_j)$ and $w_{i,j} = 0$ otherwise, i.e., assigning to each component $w_{i,j}$ the weight assigned by the weight function of the edge incident on v_i (positively) and v_j (negatively), in case such edge exists. Furthermore, the nodes' adjacency can be represented in this same matrix, by considering v_i is adjacent to v_j if and only if $w_{ij} \neq 0$.

Definition 4 (Featured graph). A featured graph is a 4-tuple (V,E,φ,xf), where (V,E,φ) is a graph and xf is a feature function $xf : v_i \rightarrow x_i$ that assigns a feature x_i to each node $v_i \in V$.

A finite featured graph can be represented by the adjacency matrix of graph (V,E,φ) and a $|V|$-sized feature vector $X = [x_1, x_2, \ldots, x_{|V|}]$.

Remark. A graph can be simultaneously *weighted* and *featured*, defined as a 5-tuple (V,E,φ,wf,xf), and as such it can be represented by its weight matrix W and its feature vector X if it is finite.

Definition 5 (Line Graph). The *line graph* (or edge-to-vertex dual graph) of a given graph $G = (V,E,\varphi)$, denoted $L(G)$, is the graph $(LV, LE, L(\varphi))$, such that $LV = \{v(e_i) : e_i \in E\}$, $LE = \{(e_i, v, e_j) : \exists u, u' \in V, \varphi(e_i) = (u,v) \wedge \varphi(e_j) = (v,u')\}$ and $L(\varphi)(e_i, v, e_j) = (e_i, e_j)$, i.e., two vertices $v(e_i), v(e_j)$ of $L(G)$ are adjacent if and only if there exists a vertex v of G on which e_i is negatively incident and e_j is positively incident.

Remark. The elements of LV denoted as $v(e_i)$ are only syntactical terms inspired in [17].

Definition 6 (Line Graph of a Weighted Graph). The line graph of a given weighted graph $G = (V,E,\varphi,wf)$ is the featured graph $L(G) = (LV, LE, L(\varphi), L(wf))$, where $(LV, LE, L(\varphi))$ is the line graph of (V,E,φ), and $L(wf)$ is the feature function $L(wf)(v(e_i)) = wf(e_i)$.

Definition 7 (Line Graph of a Featured Graph). The line graph of a given featured graph $G = (V,E,\varphi,xf)$ is the weighted graph $L(G) = (LV, LE, L(\varphi), L(xf))$, where $(LV, LE, L(\varphi))$ is the line graph of (V,E,φ), and $L(xf)$ is the weight function $L(xf)(e_i, v, e_j) = xf(v)$.

3 Graph Representation Learning

Since graphs enriched with node information are complex and difficult structures to process or compute, learning low-dimensional vector representations for graphs has attracted much interest [10, 11, 5], especially when graphs retain important properties or features. This can be used to analyse graphs or as input to other algorithms, resulting in better performance and lower cost.

Learning efficient representations for structured data is not an easy task. In [10], the authors describe several successful models that have been developed for specific types of structured data. A clear example is sequential data, such as text or video. These have been modelled by recurrent neural networks [19], which can capture sequential information and produce efficient representations that have been verified in automatic translation and speech recognition tasks. Another example is convolutional neural networks (CNN) [20] based on structural criteria, which have achieved excellent performance in pattern recognition tasks such as image classification or speech identification.

While these models have been very successful in practice, they have been limited to certain types of data, such as data with a simple relational structure, sequential data, or data that follows regular patterns. In many contexts, however, data tends to follow complex relational structures. This is the case for the graphs we have presented [21]. For structured data in graphs, defining neural networks and other computational intelligence algorithms is challenging because the input structures can be arbitrary and vary significantly between different graphs, even for nodes within the same graph. Unlike images, audio and text, which have a clear lattice structure, graphs have irregular structures. This makes it difficult to generalise some mathematical operations on graphs. For example, it is not easy to define convolution and pooling operations, which are fundamental in convolutional neural networks (CNNs), for graph data. By analogy with images, each pixel has the same neighbourhood structure, and the same weight filters can be applied to multiple locations of the image being processed. In graphs, however, each node may have a different neighbourhood structure. This problem is known as the geometric problem of deep learning [22].

Given these challenges and the widespread use of graphs in real-world applications, there has been an increased interest in applying learning methods to structured data in graphs. As a result, Graph Representation Learning (GRL) methods have emerged [10]. They aim to learn low-dimensional continuous vector representations, also called Embeddings, for structured data in graphs.

In general, GRL methods can be divided into two classes of learning problems: unsupervised and supervised GRL. The first family aims to learn low-dimensional Euclidean representations that preserve the structure and features of an input graph. The second family also learns low-dimensional Euclidean representations, but for a specific task of subsequent prediction, such as node or graph classification. They do this on data labelled for this purpose.

As defined in [10], Graph Embedding is a task that aims to learn a mapping from discrete graphs to a continuous domain. Formally, given a collection of n weighted graphs $\{G_i\}_{i=1}^{n}$, with each $G_i = (V_i, E_i, \varphi_i, wf_i)$, the goal is to learn a low-dimensional vector representation Z_i (embeddings) for each graph G_i, such that important graph properties are preserved in the embedding space. For example, if two graphs are similar according to their original representation, their learned vector representations should be also be similar. Let $Z \in \mathbb{R}^{n\times d}$ denote the graph embedding matrix. In practice, we often want low-dimensional embeddings ($d \ll \max\{|V_i|\}_{i=1}^{n}$) for scalability reasons. That is, graph embedding can be seen as a dimensionality reduction technique for graph structured data, where the input data is defined on a non-Euclidean, high-dimensional, discrete domain.

It is worth noting that, although we present the embedding task as a graph embedding task, it can also be extended to node tasks and edge tasks.

As it was defined for featured graphs, they can have node attributes (e.g. demand or availability of merchandise in logistics problems; content of articles in citation networks), commonly called node features. If there is more than one feature, they can be represented as $X \in R^{|V|\times xd}$, where xd is the number of features stored in each node. Node features can provide useful information about a graph. Some graph embedding algorithms exploit this information by learning mappings:

$$(W,X) \rightarrow Z$$

Note that depending on whether node features are used in the embedding algorithm, the learned representation may capture different aspects of the graph. Incorporating node features in embeddings allows both structural and semantic information to be captured from the graph. If node features are not used, the embeddings will only preserve the structural information of the graph.

3.1 Types of Graph Embedding

The result of graph embedding is a vector or set of low-dimensional vectors representing a graph or part of it. As described in [23], different types of graph embeddings can be classified according to the granularity of the vector set and the interpretation of what it represents. Each type of embedding enables different applications. The categories are:

- **Node Embedding**: This is the most common configuration for embedding outputs and represents each node as a vector in a low-dimensional space. Nodes that are close together in the graph are embedded to have similar vector representations.

- **Edge Embedding**: Intended to represent an edge as a low-dimensional vector, edge embedding is useful for knowledge graphs [24] and tasks related to links between nodes. It is particularly useful for analysing graphs related to edges, such as link prediction, entity and relationship prediction in knowledge graphs, and more.

- **Hybrid Embedding**: It is the combination of different components of the graph, such as nodes + edges (substructure) or nodes + communities [25], into a single embedding.

  Two additional challenges arise in this context: how to generate the subgraphs or communities to be embedded, and how to deal with the heterogeneity of the embedding targets (i.e. nodes, edges, subgraphs and communities can be embedded simultaneously).

- **Whole Graph Embedding**: whole graph embedding is typically used for small graphs, where each graph is represented as a vector and similar graphs are embedded side by side in the output vector space. This approach benefits the graph classification task by providing a direct and efficient solution for computing similarities between graphs [26].

  However, it presents challenges in capturing the properties of a complete graph and finding a balance between the expressiveness of the learned embedding and the efficiency of the embedding algorithm [26].

Finding an efficient transformation allows:

- **Preserve relevant information**: Preserve the intrinsic information of the structure and relevant properties by reducing noise or redundant information. The selection or prioritization of the information to be preserved, as previously discussed, is a critical aspect of the process.

- **Dimensionality reduction**: Discover graph representations in a lower-dimensional space.

- **Dimension standardization**: In certain applications, input graphs can have a varying dimension, or a singular graph may change dimensions with time. For many analysis tasks, particularly for their use as input in neural networks and reinforcement learning algorithms, having a dimension representation that is standardized and unvarying over time becomes necessary.

- **Enhancing computational performance and reducing costs**: The transformation to a lower dimensional vector space makes it possible to reduce the complexity of many iterative or combinatorial problems in network analysis, and to solve them using mapping functions, distance metrics or operations on embeddings.

  Eliminating coupling between nodes enables the utilization of high-performance computing solutions for large-scale network analysis.

- **Application of Computational Intelligence Algorithms**: The vector representation can be used as input in Computational Intelligence operations and algorithms that only accept matrix data, like neural networks, genetic algorithms, and others.

In summary, graph embedding is a robust technique that can enhance the precision and efficiency of machine learning algorithms in graph-related tasks.

4 Graph Representation Learning for Reinforcement Learning

Reinforcement Learning (RL) is a emerging area of research that requires the application of graph analysis as a subsidiary task.

Several studies have combined RL techniques with graph representation learning for analysis or mining tasks. These works can be classified into two categories [27]: (1) Solving RL problems using graph structures, and (2) Solving graph mining tasks with RL methods.

The second category is referred to as Graph RL in [27], and should not be confused with Graph Representation Learning (GRL). Graph RL uses RL techniques to generate embeddings. In contrast, GRL uses graph representations as inputs or as models to implement and solve RL algorithms.

Our work belongs to the first category. It aims to generate graph embeddings to be used as input for RL algorithms to solve the underlying problems.

As described by [13] and [14], RL methods have mainly been applied to simple decision-making problems, primarily related to game solving, using states represented with fixed-dimensional matrices derived from image or sensor processing, and basic decisions.

The challenge ahead is to adapt actor-critic methods to address a wide range of real-world problems that hold scientific and social significance. Reinforcement learning (RL) has the potential to enhance the quality, efficiency, and cost-effectiveness of important processes such as education, healthcare, transportation, and energy management. To achieve this, it is crucial to address the design decisions and adjustments involved in RL implementation. The architecture must be designed by selecting appropriate learning algorithms, state and action representations, training procedures, hyperparameter settings, and other design details as mentioned in[14].

The agent's state or observation is a crucial component to design and incorporate in these models. In problems of reinforcement learning where a network is present in the agent's environment, including the graph that describes the network or a fitting representation of it in the agent's observation holds a significant promise. The inclusion of the graph or a representation can enhance the agent's environment perception and, as a result, accelerate and improve its learning.

4.1 Specific requirements

RL is considered a new class of graph analysis tasks due to its distinctive characteristics and requirements compared to classic tasks mentioned previously. It is highly important to identify appropriate transformation algorithms for this specific task. It is essential to obtain a proper representation of each whole graph, which enables its utilization as input in neural networks and other computational intelligence algorithms that constitute the RL agents [13, 14, 15].

Consequently, graph embedding for implementation in RL algorithms requires:

1. **Unsupervised GRL**: data in reinforcement learning tasks are unlabelled. The objective is to create a representation based solely on the structure of each network and the relevant features to be kept. As a result, unsupervised embedding algorithms are required.
2. **Whole Graph**: the need is for a representation of the graph as a whole. Thus, algorithms that rely on embedding individual nodes, edges or substructures are not appropriate.
3. **Preserves structure and characteristics**: preserving the network structure and relevant properties of each node and/or edge is essential. Choosing which characteristics to preserve is a critical part of the process.
4. **Fixed dimension**: In the process of modelling environments for reinforcement learning, it may be necessary to represent multiple graphs with varying dimensions or a single graph that changes dimensions over time. To employ these graphs as input for neural networks and reinforcement learning algorithms, it is imperative to have a fixed-dimension representation that remains constant over time. Therefore, algorithms that produce fixed-dimensional embeddings are required.

5 A Framework for Evaluating Graph Embedding Algorithms

A universally applicable metric for quantifying the performance of a graph embedding algorithm or the quality of the reduced graph does not exist. [28] When validating or comparing graph embedding operators and algorithms, researchers typically achieve this by reconstructing the original graph or by assessing their performance on subsequent analysis tasks. [28] Typically, these analyses only examine classical graph analysis tasks, including node or graph classification, node clustering, and link prediction.

In this context, an internal validation of the algorithms is proposed based on a similarity hypothesis, i.e., if two graphs are similar when compared using their typical high-dimensional discrete representation, their low-dimensional continuous representations should also be similar. To test this, we assess the degree of ”conservation” of a distance function defined on the initial graphs. This distance function can be custom defined to prioritize the features to be preserved, both topological and attribute features of each node. Subsequently, this validation can serve as a metric to compare different algorithms.

It is a fact that it is feasible to conduct empirical tests on the efficacy of a specific embedding in a task featuring an external validation scheme. Nevertheless, all contender embedding algorithms ought to undergo these external validation tests. The duration of time spent on each embedding algorithm is already significant, and it is also susceptible to potential training errors such as underfitting, overfitting, and a lack of regularisation. The aforementioned statement holds particularly true in RL. Unlike more traditional assignments such as classification or regression, adequate algorithms are not yet available to counter these potential deficiencies. The suggested framework is evidently applicable and modifiable for any type of embedding and task.

In summary, the proposed approach is an abstract framework for validating and comparing graph embedding algorithms. The steps involved are:

1. **Definition of the Graph Set and Relevant Features**: The set of input graphs and their corresponding attributes or features to preserve should be clearly defined. This definition will depend on the particular problem to be solved or the subsequent analysis task, therefore it is important to provide clear and specific detail.
2. **Distance Function Definition**: A distance function needs to be established for the original representation of graphs (in terms of the properties to be preserved), as well as for the vector representations that are generated by the algorithms. The most commonly used distance metric for the transformed space is the Euclidean measure, although it is not the only one.
3. **Algorithm selection**: The selection of embedding algorithms that meet the requirements defined in step one for a specific task or problem is necessary.
4. **Hyper Parameter Definition**: Hyperparameters should be defined for the selected algorithms, particularly the embedding dimension (d).
5. **Embedding Generation**: Each model must be trained by each algorithm and then calculate the graph embeddings for each of the trained algorithms.
6. **Distance Calculation**: Using the functions defined, the distances between graphs must be calculated in all representations.

7. **Correlation Analysis**: To perform the internal validation of the algorithms, based on the conservation of the distance function defined on the original graphs, a correlation analysis with the distances between the corresponding embedding vectors is proposed. This analysis aims to confirm the correlation between the distances calculated from the original representation and the distances calculated from the embeddings, also evaluating the strength of that correlation. A stronger correlation between these distance distributions implies that the preservation of the initial distance in the learned representation is achieved, meaning that the most remote or distinct graphs in the original discrete domain are also distant in the created continuous space.

It is recommended to utilize both Pearson's correlation coefficient r (parametric) and Spearman's correlation coefficient ρ (non-parametric) to strengthen the analysis's robustness. If the Pearson correlation coefficient is close to 1, then the distances are linearly correlated, implying a certain proportionality between the distances of the two spaces. If the Pearson coefficient approaches -1, the transformation will invert distances in a linear fashion, resulting in distant graphs in one space appearing close in the other, and vice versa. If Pearson's coefficient equals 0, then distances are not linearly correlated, though non-linear correlation may still exist. Therefore, it is recommended to also use Spearman's rank correlation coefficient, which is non-parametric and can be interpreted similarly to Pearson's, complementing and strengthening the analysis.

6 Experimentation

Following the proposed evaluation framework and addressing a specific problem, three algorithms were selected, analyzed, and compared: Graph2Vec, GL2Vec, and Wavelets Characteristics. All three algorithms meet the necessary requirements described above.

The tests were coded in Python using the NetworkX [29] and Karate Club [30] libraries. The source code can be found at [31].

6.1 Graph set and relevant feature definition

The collection of graphs to be employed comprises examples of the vehicle routing problem (VRP) that can be solved through RL. VRP is a more extensive type of the Travelling Salesman Problem (TSP) that considers multiple vehicles in its routing model. There is a fleet of identical vehicles available to serve a set of geographically scattered customers centered around a primary depot [32, 33]. Therefore, VRP is concerned with optimal service provision to all its customers. Since its formulation in 1959, VRP modelling has been extensively studied in the context of operational research, industrial engineering, logistics, and transportation.

To model this problem, a weighted and featured graph is used, where each node represents a location on the future route (including customers and the central depot), and the edges are the travel times between each of these locations. Each graph is represented by a weighted matrix W that represents the travel times at each weight, and a feature vector X that represents the customer's demands.

6.2 Distance Functions Definition

The distance between graphs in their original representation is defined by the *graph edit distance*, a definition of which can be found in [34]. The strategy that follows is similar to Levenshtein's edit distance for strings, where a graph is subjected to replacements, insertions or deletions of nodes or edges in order to obtain another graph that proves to be isomorphic to the second one. Although this metric originally focuses on the topology of the graph, it is possible to parameterise it using differential costs for insertions, deletions and replacements.

Given two arbitrary weighted featured graphs $G_1 = (V_1, E_1, \varphi_1, wf_1, xf_1)$ and $G_2 = (V_2, E_2, \varphi_2, wf_2, xf_2)$, the costs of the edit operations $cost(op_{obj})$, where $op \in \{subst, del, ins\}$ and $obj \in \{node, edge\}$, are defined as follows

$$cost(subst_{node}(v_1 \in V_1, v_2 \in V_2)) = |xf_1(v_1) - xf_2(v_2)|$$
$$cost(del_{node}(v_1 \in V_1)) = |xf_1(v_1)|$$
$$cost(ins_{node}(v_2 \in V_2)) = |xf_2(v_2)|$$
$$cost(subst_{edge}(e_1 \in E_1, e_2 \in E_2)) = |wf_1(e_1) - wf_2(e_2)|$$
$$cost(del_{edge}(e_1 \in E_1)) = |wf_1(e_1)|$$
$$cost(ins_{edge}(e_2 \in E_2)) = |wf_2(e_2)|$$

Also, the metric used to calculate the distance between embeddings is the Euclidean distance.

6.3 Selected Algorithms

As stated previously, all three algorithms meet the essential criteria outlined earlier. Each one is:

Graph2Vec This algorithm is detailed in [16]. It utilises word and document embedding techniques introduced in the area of Natural Language Processing [35]. The algorithm undertakes an analogy wherein an whole graph is considered as a document and the rooted subgraphs (spanning a neighbourhood of a certain degree) around each graph node are the words that make up the document. By utilizing the analogy of documents and words with graphs and subgraphs, document embedding models can be used to learn graph embeddings. Given a dataset of graphs, Graph2Vec considers the set of all rooted subgraphs (the neighbourhood) around each node (up to a certain degree) as a vocabulary. Subsequently, the algorithm proceeds with the Doc2Vec skip-gram training procedure [36] to autonomously acquire the representations of each graph in the dataset.

GL2Vec This algorithm, proposed in [17], aims to improve Graph2Vec by addressing its two limitations: (1) it cannot handle the labels of the edges, and (2) structural information is not properly preserved, since Graph2Vec mixes information from node labels with structural information when extracting subgraphs. To address these limitations, the authors suggest utilizing the line graph (a dual edge-to-vertex graph) of G. The nodes of L(G) incorporate the properties of the edges and node labels of G, thus preserving both types of information. This approach follows Definition 6 and is ideal for handling G's structural information. Embeddings are generated for both G and L(G), and an embedding comprises the original graph embedding concatenated with that of the linear graph. This enhances the structural information of G that Graph2Vec disregards. Therefore, the technique is called GL2Vec (Graph and Line graph to vector).

Wavelet Characteristics This algorithm was proposed in [18]. The process utilises the characteristic functions of node attributes with wavelet function weights to describe node neighbourhoods. These node-level features are grouped using mean binning to create graph-level statistics.

All three selected algorithms utilize an embedding dimension of 128, and their corresponding hyperparameters can be located in the code.

7 Experimental Results

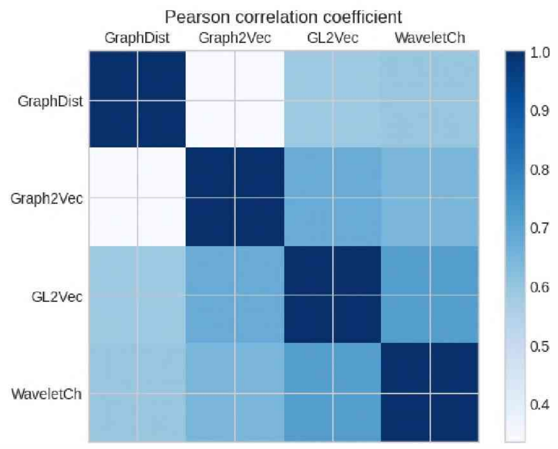
For each chosen algorithm, the model was trained, and the graph embeddings were calculated. In all instances, the functions outlined in Section 6.2 were utilized to compute the distances between the graphs and embeddings. In addition, the results were analysed using both Pearson's and Spearman's correlation coefficients.

The correlation analysis of the obtained results is displayed in Table 1, whereas Figure 1 depicts the graphical representation of both correlation coefficients. It is noted that both correlation coefficients show a similar trend in all cases.

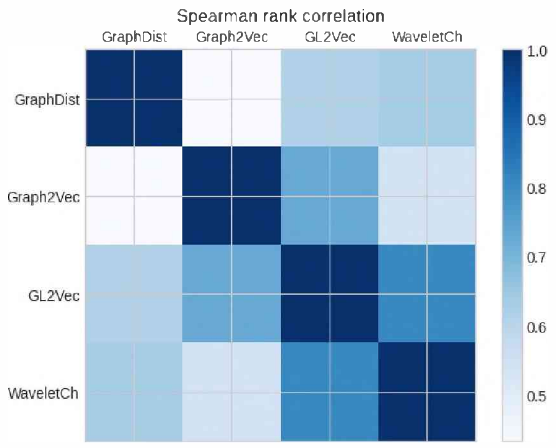
Table 1: Correlation analysis results

	Graph edit dist. vs. Euclidean Distance for Embeddings using:		
	Graph2Vec	**GL2Vec**	**Wavelet**
Pearson's r	0.3349	0.5826	0.5970
Spearman's ρ	0.4336	0.6087	0.6337

The results indicate that the Wavelet Characteristics algorithm had the highest correlation coefficients, achieving a Pearson coefficient of 0.597 and a Spearman coefficient of 0.63. GL2Vec yielded satisfactory results, exhibiting correlation coefficients of 0.58 and 0.608 for Pearson and Spearman, respectively. Its performance surpassed that of Graph2Vec, aligning with the theoretical framework suggested by its creators [17].



(a) Pearson



(b) Spearman

Figure 1: Pearson and Spearman Correlation Coefficients

8 Conclusions and Future Works

This paper presents a framework for evaluating and comparing Graph Embedding algorithms, emphasising their aptitude to retain structure and relevant features through internal validation. It outlines the motivation and introduces the fundamental concepts of Graph Representation Learning. Further to this, classical graph analysis tasks are specified, and reinforcement learning (RL) is highlighted as an emerging field that can benefit from graph analysis as an subsidiary task.

Following this framework, and for a specific Reinforcement Learning problem in graphs, three algorithms available in the state of the art that meet the defined requirements have been examined. Especially, the Wavelet Characteristics and GL2Vec algorithms have yielded promising results.

As a future work, we plan to implement an RL Agent for the VRP problem defined by utilizing the algorithms that demonstrate the best correlation. This will enable external validation of the algorithms in the specific task. It is also planned to extend the analysis to graphs containing fuzzy data, including the modelling of demand and travel times using fuzzy numbers. By using fuzzy numbers, it will be possible to incorporate uncertainty and generate more robust and adaptive models. Finally, another opportunity is to utilise Graph Embedding to represent actions in RL.

Competing interests

The authors have stated that they do not have any competing interests.

Authors' contribution

The authors confirm contribution to the paper as follows: ES: Conceptualization, Methodology, Data curation, Investigation, Software, Visualization, Validation, Writing-Reviewing and Editing; CC: Conceptualization, Methodology, Supervision, Validation, Writing-Reviewing and Editing; FP: Validation, Writing-Reviewing and Editing. All authors reviewed the results and approved the final version of the manuscript.

References

[1] L. Freeman, "Visualizing social networks," *Journal of social structure*, vol. 1, no. 1, p. 4, 2000.

[2] R. Ferrer I Cancho and R. Solé, "The small world of human language," *Proc. Biological sciences*, vol. 268, no. 1482, p. 2261—2265, November 2001. [Online]. Available: https://doi.org/10.1098/rspb.2001.1800

[3] T. Theocharidis, S. van Dongen, A. Enright, and T. Freeman, "Network visualization and analysis of gene expression data using biolayout express(3d)," *Nature Protocols*, vol. 4, no. 10, pp. 1535 –1550, 2009. [Online]. Available: https://doi.org/10.1038/nprot.2009.177

[4] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. on Knowl.Discovery from Data*, vol. 1, no. 1, pp. 2–43, 2007. [Online]. Available: https://doi.org/10.1145/1217299.1217301

[5] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018. [Online]. Available: https://doi.org/10.1016/j.knosys.2018.03.022

[6] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc, of the 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, USA: Assoc. for Computing Machinery, 2014, p. 701–710. [Online]. Available: https://doi.org/10.1145/2623330.2623732

[7] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, USA: Assoc. for Computing Machinery, 2016, p. 1105–1114. [Online]. Available: https://doi.org/10.1145/2939672.2939751

[8] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proc. 2001 IEEE Int. Conf. on Data Mining*, 2001, pp. 107–114. [Online]. Available: https://doi.org/10.1109/ICDM.2001.989507

[9] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, USA: Assoc. for Computing Machinery, 2016, p. 1225–1234. [Online]. Available: https://doi.org/10.1145/2939672.2939753

[10] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *Journal of Machine Learning Research*, vol. 23, no. 89, pp. 1 –64, 2022. [Online]. Available: http://jmlr.org/papers/v23/20-852.html

[11] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 5, pp. 833 –852, 2018.

[12] C. L. Staudt, A. Sazonovs, and H. Meyerhenke, "Networkit: A tool suite for large-scale complex network analysis," *Network Science*, vol. 4, no. 4, pp. 508 –530, 2016.

[13] R. Sutton and A. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[14] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Looking back on the actor–critic architecture," *IEEE Trans.on Syst., Man and Cybernetics: Systems*, vol. 51, no. 1, pp. 40–50, 2021. [Online]. Available: https://doi.org/10.1109/TSMC.2020.3041775

[15] E. Schab, C. Casanova, and F. Piccoli, "Solving an instance of a routing problem through reinforcement learning and high performance computing," in *Cloud Computing, Big Data & Emerging Topics*, E. Rucci, M. Naiouf, F. Chichizola, L. De Giusti, and A. De Giusti, Eds. Cham: Springer Int. Publishing, 2022, pp. 107 –121.

[16] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," 2017.

[17] H. Chen and H. Koga, "Gl2vec: Graph embedding enriched by line graphs with edge features," in *Neural Information Processing*, T. Gedeon, K. W. Wong, and M. Lee, Eds. Cham: Springer Int. Publishing, 2019, pp. 3 –14. [Online]. Available: https://doi.org/10.1007/978-3-030-36718-3_1

[18] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," 2020.

[19] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015.

[20] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022. [Online]. Available: https://doi.org/10.1109/TNNLS.2021.3084827

[21] A.-L. Barabási, "Network science," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1987, p. 20120375, 2013.

[22] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2693418

[23] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge & Data Engineering*, vol. 30, no. 09, pp. 1616–1637, sep 2018. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2807452

[24] A. Bordes, X. Glorot, J. Weston, and Y. Bengio, "A semantic matching energy function for learning with multi-relational data: Application to word-sense disambiguation," *Machine Learning*, vol. 94, pp. 233 –259, 2014.

[25] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proc. of the AAAI Conf. on AI*, vol. 31, no. 1, 2017.

[26] S. F. Mousavi, M. Safayani, A. Mirzaei, and H. Bahonar, "Hierarchical graph embedding in vector space by graph pyramid," *Pattern Recognition*, vol. 61, pp. 245–254, 2017.

[27] M. Nie, D. Chen, and D. Wang, "Reinforcement learning on graphs: A survey," 2023.

[28] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022. [Online]. Available: https://doi.org/10.1109/TNNLS.2022.3190922

[29] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proc. of the 7th Python in Science Conf.*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[30] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in *Proc. of the 29th ACM Int. Conf. on Information and Knowledge Management (CIKM '20)*. ACM, 2020, p. 3125–3132.

[31] E. A. Schab, "estebanschab/RL-GraphEmbeddings: Release for publication in JCC2023," Apr. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7830059

[32] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, no. 4, pp. 568 –581, 1964. [Online]. Available: http://www.jstor.org/stable/167703

[33] M. Asghari and M. Mirzapour Al-e-hashem, "Green vehicle routing problem: A state-of-the-art review," *Int. Journal of Production Economics*, vol. 231, p. 107899, 2021. [Online]. Available: https://doi.org/10.1016/j.ijpe.2020.107899

[34] B. Jain and K. Obermayer, "Graph quantization," *Computer Vision and Image Understanding*, vol. 115, no. 7, pp. 946–961, 2011. [Online]. Available: https://doi.org/10.1016/j.cviu.2011.03.004

[35] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Int. Conf. on machine learning*. PMLR, 2014, pp. 1188 –1196.

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

Citation: E. Schab, C. Casanova and F. Piccoli. *Graph Representations for Reinforcement Learning.* Journal of Computer Science & Technology, vol. 24, no. 1, pp. 29-38, 2024.
DOI: 10.24215/16666038.24.e03
Received: April 15, 2023 **Accepted:** January 10, 2024.