IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 May 2024, accepted 12 June 2024, date of publication 17 June 2024, date of current version 15 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3415742

RESEARCH ARTICLE

Flexible Graph Comparison Using HMMs

MOHAMMAD MOURAD ABDOULAHI[1] **AND SYLVAIN ILOGA**[2,3,4,5]

[1]Faculty of Sciences, Department of Mathematics and Computer Science, University of Maroua, Maroua, Cameroon
[2]Higher Teachers' Training College, Department of Computer Science, University of Maroua, Maroua, Cameroon
[3]ENSEA, CNRS, ETIS UMR 8051, CY Cergy Paris University, 95000 Cergy, France
[4]ESIEE-IT, CCI Paris île-de-France, 95300 Pontoise, France
[5]CIRD, UMMISCO, University of Sorbonne, 93143 Bondy, France

Corresponding author: Sylvain Iloga (sylvain.iloga@gmail.com)

This work was supported by école d'ingénieurs et de l'expertise numérique (ESIEE-IT), Chambre de Commerce et d'Industrie (CCI) Paris île-de-France.

ABSTRACT Graphs are powerful means for representing structured data. Graph comparison is consequently an important tool for decision support and several techniques have therefore been proposed for comparing two graphs, including Substructure-based techniques, Matrix-based techniques, Graph Kernels and Graph Neural Networks. In addition to their individual limitations, all these existing techniques do not allow the explicit specification of criteria on which the comparison should rely and which intrinsically determine the result of the comparison. The current paper attempts to attenuate the drawbacks of the existing techniques for graph comparison by proposing a flexible technique based on hidden Markov models (HMMs) that offers the possibility to explicitly precise a finite set of properties that must be considered during the comparison. The proposed approach transforms each graph into a set of Markov chains that is later used for initializing, then for training several HMMs. An interpretable descriptor vector associated with each graph is then extracted from the HMMs. The comparison between two graphs is finally performed through the comparison of their corresponding descriptor vectors by using any existing vector distance. Classification experiments conducted on six graph datasets including two real-world datasets demonstrated that, when the suitable set of properties is selected, the proposed approach outperforms existing techniques with accuracy gains reaching +54.38%.

INDEX TERMS Graph, graph comparison, graph modeling, hidden Markov models.

I. INTRODUCTION

Graphs are powerful means for representing structured data in complex systems because they represent both, entities (vertices) and relationships between these entities (edges). Consequently, the amount of data naturally modeled as graphs has significantly increased since many years. Graph data have therefore become ubiquitous in several application domains including (but not limited to) biology, chemistry and social networks [1]. A more detailed list of application domains where graph-structured data are used is available in [2].[1] Protein-protein interaction networks, metabolic networks, regulatory networks and phylogenetic networks are examples of graph-structured data that can be found in biology. In chemistry, molecular compounds are generally modeled as graphs where vertices are atoms and edges are chemical bonds. A social network is a graph where individuals are represented as vertices, while the interactions (friendship, collaboration, etc.) between them are represented as edges. The use of graphs can be extended to data that do not inherently have an underlying graph structure. This is the case for sequential data as text that can be mapped to graph structures [3].

Graph-structured data can be manipulated by different operations including graph comparison, which is an important tool for decision support while being crucial for popular machine learning tasks like *classification* or *clustering*. Given a dataset composed of many classes, each class containing several graphs, classification refers to the process of assigning a class label to a graph based on its properties [4]. Clustering rather consists in organizing all the graphs of a set of graphs into separated clusters (groups) with the most similarity in the same cluster and the most dissimilarity between different clusters [5]. Graph comparison is also essential for the analysis of graph-based systems where each graph represents the state of the system at specific point in time or space.

The associate editor coordinating the review of this manuscript and approving it for publication was Khursheed Aurangzeb.

[1]See Table 3 of [2].

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

Indeed, the analysis of the dynamics of such systems depends on how much the analytical tool is capable to reflect the similarities between the states (graphs) [6].[2]

Several techniques have yet been proposed for comparing two graphs. These techniques can be organized into four main categories: *substructure-based techniques* (SSBTs) [7], [8], [9], [10], [11], [12], [13], *matrix-based techniques* (MBTs) [6], *graph kernels* (GKs) [1], [14] and *graph neural networks* (GNNs) [2], [15], [16]. SSBTs rely on element-to-element comparisons between two graphs, the elements being the graph substructures (vertices, edges, subgraphs, etc) of the two graphs. But they are limited by their high computational complexity. MBTs are adapted for the study of network dynamics and exclusively compare the graphs through the manipulation of their associated matrices (*adjacency matrix*, *Laplacian*, etc.). As a consequence, they can only compare two aligned graphs, these are two graphs defined on the same set of vertex identifiers [6].[3] This problem is solved by GKs which measure the similarity between pair of graphs, irrespective of their number of vertices. Roughly speaking, GKs embed the graphs into vector spaces by using a deterministic mapping function before performing the comparison. Therefore, GKs can capture as much as possible the semantic inherent in the graphs, while remaining computationally efficient. The major limitation of GKs is that each kernel only prioritizes a predefined and limited set of graphs properties. GNNs represent a more efficient alternative than GKs for graph comparison [16].[4] They also embed the graphs into vector spaces by using a mapping function, but for GNNs, the mapping function is iteratively learnable directly from the graph structure. GNNs can be further empowered by the integration of deep models. Therefore, they can compare any two graphs, irrespective of their properties. Unfortunately, GNNs suffer form the following drawbacks inherited from deep models listed in [17][5]: they require lots of computing resources, the components of their generated feature vectors are less interpretable/explainable and the hyper-parameters of the deep models are difficult to adjust.

Moreover, all the aforementioned existing techniques do not allow the explicit specification of user-defined criteria on which the comparison should be performed. However, comparison is fundamentally a flexible operation because the result $d(x, \overline{x})$ of the comparison between two items x and $\overline{x}$ according to a metric d is not unique, but rather depends on the set P of comparison criteria. For example, consider the two balls x and $\overline{x}$ respectively depicted in Figures 1a and 1b. When $P = \{shape, brand\}$, the comparison result is $d(x, \overline{x}) =$ *'similar'* because they are both spheres designed by the '*Nike*' brand. But the opposite result $d(x, \overline{x}) =$ *'different'* is obtained for these same balls when $P = \{volume, texture, color, usage\}$ because x is smaller than $\overline{x}$ and, x is a smooth blue ball used for playing football, while $\overline{x}$ is a rough brown ball with bumps used for playing basketball. Although these two comparison results are different, they both remain fully consistent according to their respective sets of comparison criteria. Hence, the comparison between two graphs can also rely on user-defined criteria verified by their vertex/edge characteristics. As an example, one may want to compare two molecules according to the *parity* of the number of chemical bonds attached to each atom, that may have a particular importance in a specific experimental context.

(a) The ball x (b) The ball $\overline{x}$

FIGURE 1. Two balls compared according to different criteria.

This paper attempts to attenuate the limitations of the existing techniques for graph comparison by proposing a flexible technique that also embeds the graphs into vector spaces using a learnable mapping function like GNNs, but that unlike existing techniques, additionally offers the possibility to explicitly precise a user-defined set P of properties that must be considered during the comparison. More formally, given a graph x and a user-defined finite set $P = \{p_1, \ldots, p_m\}$ of properties, a hidden Markov model (HMM) $\lambda_i(x)$ is initialized then trained for capturing the adherence of the vertices/edges of x to the property $p_i \in P$ with $(1 \leq i \leq m)$. Thereafter, metadata derived from $\lambda_i(x)$ are saved as the components of a feature vector $\mu_i(x)$. When this process is applied for all the m properties in P, the set $\{\mu_1(x), \ldots, \mu_m(x)\}$ is obtained and subsequently used for deriving a unique feature vector $\mu_P(x)$ associated with x according to the properties in P. The distance/similarity between two graphs x and $\overline{x}$ is finally computed by comparing their respective feature vectors $\mu_P(x)$ and $\mu_P(\overline{x})$ using any existing distance/similarity between vectors. The performances of the proposed approach have been evaluated through classification experiments on six graph datasets including two real-world datasets from the *TUDataset* repository [18],[6] which is a collection of benchmark datasets for learning with graphs.

As it will be further discussed in Section V-D, the main contributions of the current paper are:

1) It performs graph comparison according to a finite user-defined set of properties using HMMs. No existing technique offers this flexibility.
2) The training time of these HMMs is reasonable and their parameters can be easily modified.
3) It associates an interpretable descriptor vector with each graph.

[2] See the Abstract of [6].
[3] See Section 1.2 of [6].
[4] See the last paragraph of Section II-A of [16].
[5] See Section II-B of [17].
[6] https://chrsmrrs.github.io/datasets/docs/datasets/

TABLE 1. Main parameters used in the paper.

Parameter	Description
x,$\overline{x}$	Graphs
$d_H(x,\overline{x})$	Hamming distance
$d_J(x,\overline{x})$	Jaccard distance
$d^f_{LP}(x,\overline{x})$	l_p distance using a function f
$d_{ST}(x,\overline{x})$	Spanning tree dissimilarity
$d_{IM}(x,\overline{x})$	Ipsen-Mikhailov distance
$d_G(x,\overline{x})$	l_1 distance using a Gaussian kernel
$d_{HIM}(x,\overline{x})$	Hamming-Ipsen-Mikhailov distance
$\phi(x)$	Mapping function for graph kernels
$K(x,\overline{x})$	Graph kernel
$d_{KI}(x,\overline{x})$	Kernel metric
$d_{KE}(x,\overline{x})$	Distance using an explicit graph kernel
$\theta(x)$	Mapping function for GNNs
$d_\theta(x,\overline{x})$	Distance using GNNs
P	Set of user-defined properties
β	User-defined constant representing the level of neighborhood to be considered in the method
$\mathcal{G}(u)$	Subgraph explored by DFS starting from the vertex u
$\delta_i(u)$	Markov chain derived from the proposed method when the property is p_i and the DFS starts from the vertex u
$\lambda_i(x)$	HMM that captures the adherence of the vertices/edges of the graph x to the property $p_i{\in}P$ $(1{\leq}i{\leq}m)$
$\mu_i(x)$	Feature vector containing the meta-data derived from $\lambda_i(x)$ $(1{\leq}i{\leq}m)$
$\mu_P(x)$	Final feature vector associated with the graph x according to the properties in P
$d_P(x,\overline{x})$	Distance based on the proposed method

4) It handles all kinds of graphs (small, large, dense, etc.) because it considers the local environment of each vertex, as well as the global structure of the graph.

The rest of this paper is organized as follows: the state of the art is presented in Section II, followed by a synthetic description of HMM in Section III. The proposed approach is described in Section IV, while experimental results are presented in Section V. The last section is devoted to the conclusion.

II. LITERATURE REVIEW ON GRAPH COMPARISON

A. PRELIMINARIES

We present in this section basic concepts and algorithms that will be used throughout the paper. More detailed overviews on graph theory can be found in [19] and [20] and popular platforms that provide codes for graph computing are listed in [2].[7] We start by defining a graph and some essential graph concepts. The most used graph representations are then presented, and the section ends with the description of the algorithm dedicated to the traversal of a graph that has been selected in this paper. Table 1 summarizes the main parameters used throughout the paper with their corresponding descriptions.

1) DEFINITIONS

A *graph* x is a pair (V, E) where V is a finite set of objects called *vertices* (or *nodes*) and the size of x is its number $|V|$ of vertices. $E{\subseteq}V{\times}V$ is a set of *edges*, which are connections between vertices. More formally, E is a set of pairs (u, v) such that there exists in graph x a connection between the vertices u and v. When the pairs in E are ordered (i.e, $(u, v) \neq (v, u)$), x is called a *directed graph*, otherwise x is an *undirected graph*. If the pair $(u, v){\in}E$, then v is adjacent to u. The *degree* of a vertex u in graph x noted $deg(u)$ is the number of vertices adjacent to it, more formally $deg(u) = |\{v{\in}V|(u, v){\in}E\}|$. A graph $\tilde{x} = (\tilde{V}, \tilde{E})$ is a *subgraph* of a graph $x = (V, E)$ if $V \subseteq \tilde{V}$ and $E \subseteq \tilde{E}$. If $\tilde{x}$ is a subgraph of x, then x is a supergraph of $\tilde{x}$.

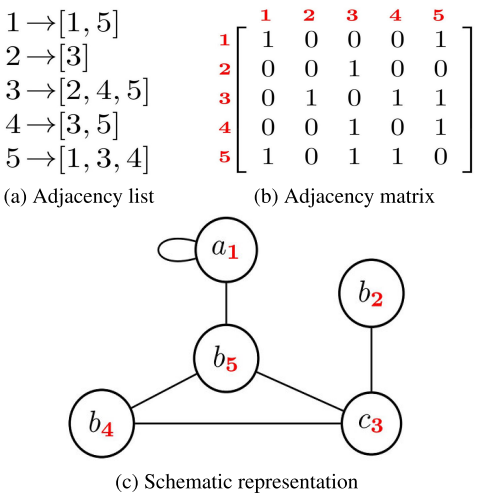
A *labeled graph* is a graph x endowed with a function $l : V{\cup}E \rightarrow \sum$ that assigns labels to the vertices and edges of x from a discrete set $\sum$ of labels. A graph with labels on its vertices is a *vertex-labeled* graph, and a graph with labels on its edges is an *edge-labeled* graph. A graph that is neither vertex-labeled, nor edge-labeled is an *unlabeled graph*.

An *attributed graph* is a graph x endowed with a function $f : V{\cup}E \rightarrow \mathbb{R}^b$ that assigns b-dimensional real vectors to the vertices and edges of x. A graph with attributes on its vertices is a *vertex-attributed* graph, and a graph with attributes on its edges is an *edge-attributed* graph. A graph that is neither vertex-attributed, nor edge-attributed is an *unattributed graph*. It is important to remark that labeled graphs are a special case of attributed graphs because their discrete labels can be mapped to one-hot vector representations.

Two unlabeled graphs $x = (V, E)$ and $\overline{x} = (\overline{V}, \overline{E})$ are *isomorphic*, denoted by $(x \simeq \overline{x})$, if there exists a bijection $h : V \rightarrow \overline{V}$ such that $(u, v){\in}E$ if and only if $(h(u), h(v)){\in}\overline{E}$ for all u, v in V. For the case of labeled graphs, isomorphism holds only if the bijection maps only vertices and edges with the same label. The *Weisfeiler-Lehman graph isomorphism test* [21] proposed in 1968 is an effective and computationally efficient test known to distinguish a broad class of graphs based on graph isomorphism.

A *path* in a graph x is a sequence of edges that connects a sequence of vertices. If x is a directed graph, the edges in the path must all be directed in the same direction. A *cycle* in a graph is a path such that the first vertex of the path corresponds to the last vertex. A graph containing at least one cycle is a *cyclic graph*, otherwise the graph is an *acyclic graph*. A *connected graph* is a graph where there exists a

[7] See Table B.6 of Appendix B in [2].



(a) Adjacency list (b) Adjacency matrix

(c) Schematic representation

FIGURE 2. **Representations of the same graph.**

path from any vertex to any other, otherwise the graph is a *disconnected graph*. A *tree* is a connected acyclic graph.

2) GRAPH REPRESENTATIONS

There are several ways for representing graphs among which we can list the three following graph representations:

1) The schematic representation (i.e: a figure describing the graph). In this representation, vertices are represented with circles, each circle carrying characteristics (identifier, label, attributes) related to the vertex it describes. In this paper, vertex identifiers are red and in bold. For vertex-labeled graphs, the vertex identifiers are placed as subscripts of the vertex labels. Edges are represented with lines for undirected graphs and with arrows for directed graphs. Each line/arrow carrying the characteristics (label, attributes) related to the edge it describes.
2) The adjacency list. It represents a graph x as an array of size $|V|$ such that the element at index u is a list containing the $deg(u)$ vertices that are adjacent to u in x.
3) The adjacency matrix. It represents a graph x as a $|V|\times|V|$ symmetric matrix A such that given two vertices u_i and u_j, the coefficient $A_{ij} = 1$ if $(u_i, u_j)\in E$, otherwise $A_{ij} = 0$.

As an example, the adjacency list presented in Figure 2a and the adjacency matrix presented in Figure 2b both correspond to the undirected, unattributed and vertex-labeled graph depicted in Figure 2c where the set of vertex labels is $\sum = \{a, b, c\}$. Vertex identifiers are bold and red in Figures 2b and 2c.

3) TRAVERSAL OF A GRAPH

To traverse a graph x means to visit all the vertices of x in some systematic order. Among the existing algorithms dedicated to the traversal of a graph, we will only focus on the *Depth-First Search* (DFS) algorithm[8] in this paper motivated by the simplicity of its implementation. DFS is a recursive algorithm that traverses x starting from each vertex $u\in V$. While traversing x starting from a vertex u, DFS manages an ordered list L_u of vertices which is empty at the beginning of the algorithm and must contain all the visited vertices of x at the end. DFS also manages another ordered list $\tilde{L}_u$ of edges which is empty at the beginning of the algorithm and must contain all the edges of x that have been visited at the end. Several versions of DFS have yet been implemented, but they all rely on the principle presented in Algorithm 1, that calls Algorithm 2. In these two algorithms, the variable *depth* refers to the number of edges in the actual path followed by DFS from the starting vertex to the currently visited vertex. The **PreUse** function refers to any user-defined task that must be executed before the exploration of the next vertex/edge by DFS. Similarly, the **PostUse** function refers to any user-defined task that must be executed just after the exploration of the currently visited vertex/edge by DFS. For each of these two functions, if no specific task needs to be executed, the function can be ignored.

Algorithm 1 starts by browsing the vertices of the graph and for each vertex $u\in V$ (line 1), it initializes the lists L_u and $\tilde{L}_u$ to the empty set (lines 2-3), meanwhile the variable *depth* is initialized to zero (line 4). The next step consists in calling Algorithm 2 for performing the traversal of the graph starting from u (line 6), this traversal is preceded (line 5) and followed (line 7) by user-defined actions performed for using u and the current value of *depth*. The algorithm finally returns all the resulting lists gathered into the sets L_x and $\tilde{L}_x$ (lines 9-11).

In Algorithm 2, the input vertex u is first inserted at the end of the ordered list L (line 1). The vertices that are adjacent to u are then collected into the set Z (line 2) and browsed (line 3). The currently visited edge is subsequently inserted at the end of the ordered list $\tilde{L}$ (line 7), before to recursively process the currently visited vertex (line 9). This recursive vertex processing is preceded (line 8) and followed (line 10) by user-defined actions performed for using the currently visited vertex/edge and the current value of *depth*. The variable *depth* is regularly updated before (line 5) and after (line 11) each recursive call. The algorithm returns the updated lists (line 14).

In a typical implementation, there is no precision related to the order in which vertices are browsed in these two algorithms (line 1 of Algorithm 1 and line 3 of Algorithm 2). For convenience, we assume in this paper that these two algorithms browse the vertices *in increasing order of the vertex identifiers*.

B. SUBSTRUCTURE-BASED TECHNIQUES

Many techniques have yet been developed to compare graphs through the comparison of their substructures (i.e: vertices, edges, subgraphs, etc). In 1970, Corneil and Gotlieb [7] first proposed to compare undirected graphs using

[8]https://jeffe.cs.illinois.edu/teaching/algorithms/book/06-dfs.pdf

Algorithm 1 DFS

Input: $x = (V, E)$
Outputs: $L_x, \tilde{L}_x$

for each $(u \in V)$ **do**
  $L_u \leftarrow \emptyset$
  $\tilde{L}_u \leftarrow \emptyset$
  $depth \leftarrow 0$
  PreUse$(depth, u)$
  DFSrec$(x, L_u, \tilde{L}_u, u, depth)$
  PostUse$(depth, u)$
end for
$L_x \leftarrow \{L_u \mid u \in V\}$
$\tilde{L}_x \leftarrow \{\tilde{L}_u \mid u \in V\}$
return $L_x, \tilde{L}_x$

Algorithm 2 DFSrec

Inputs: $x = (V, E), L, \tilde{L}, u \in V, depth$
Outputs: $L, \tilde{L}$

Insert(u, L)
$Z \leftarrow \{v \in V \mid (u, v) \in E\}$
for each $(v \in Z)$ **do**
  if $(v \notin L)$ **then**
    $depth \leftarrow depth + 1$
    $e \leftarrow (u, v)$
    Insert$(e, \tilde{L})$
    PreUse$(depth, v, e)$
    DFSrec$(x, L, \tilde{L}, v, depth)$
    PostUse$(depth, v, e)$
    $depth \leftarrow depth - 1$
  end if
end for
return $L, \tilde{L}$

a non-deterministic algorithm based on graph isomorphism, that induces vertex-to-vertex comparisons.

Many authors later proposed techniques to compare attributed graphs. It is the case of Sanfeliu and Fu [8] in 1983 for pattern recognition and Eshera and Fu [9] in 1984 for image analysis. They both compared attributed graphs by counting the number of edges present in one graph and not in the other.

In 1998, Bunke and Shearer [10] proposed a technique based on the maximal common subgraph, followed in 2001 by Fernández and Valiente [11] who performed graph comparison by combining the maximum common subgraph and the minimum common supergraph. But, their approaches failed to keep the global graph structure.

In 2003, Champin and Solnon [12] used generic functions to determine the best mapping between the vertices of directed vertex-labeled graphs. One year later 2004, Blondel et al. [13] proposed to compare directed unlabeled and unweighted graphs by measuring the similarity between their vertices for web searching purposes.

C. MATRIX-BASED TECHNIQUES

Existing MBTs between aligned graphs are organized in three main categories: *structural distances*, *spectral distances* and *mesoscale distances*.

The *structural distances* [6] capture local changes around each vertex in the graph structure. These distances are consequently suitable for comparing graph data where local changes can seriously impact the comparison result. This is for example the case of molecules where little changes in the molecule bonds can induce radically different properties (toxicity, solubility, etc.).

At the other extreme, *spectral distances* [22], [23], [24] rather capture global changes in the overall graph structure. These distances are therefore adapted for graph data where the comparison result can be fundamentally impacted by the global organization of the vertices in the graph and by their interactions. Such distances are for example better suited for comparing two brain networks where the global changes in the connectivity are important.

Mesoscale distances [6], [25] compare graphs at an intermediate scale by combining the advantages of both, the structural and the spectral distances. Such distances thus attempt to capture local changes around each vertex in the graph structure, while also considering the global changes in the overall graph structure.

1) STRUCTURAL DISTANCES

Given two graphs x and $\overline{x}$, the two most used structural graph distances are the *Hamming distance* $d_H(x, \overline{x})$[9] and the *Jaccard distance* $d_J(x, \overline{x})$.[10] The *Hamming distance* is a special instance of the well-known *graph Edit distance*, which evaluates the number of edges deletions and insertions required for transforming one graph into another. This distance is suitable for characterizing the evolution of a given system through the time, but it fails at comparing graph dynamics in the presence of different degree densities due to its dependency to the graph sparsity [6].[11] A solution to this limitation is offered by the *Jaccard distance*, which includes in its normalization, the average sparsity of the two graphs.

2) SPECTRAL DISTANCES

The ℓ_p *distance* $d^f_{Lp}(x, \overline{x})$[12] between x and $\overline{x}$ is defined as the ℓ_p distance between functions of their eigenspectra,[13] for any almost everywhere differentiable user-defined function f. One can also evaluate the number of spanning trees that are destroyed or created by the transformation of one graph to another by computing their *spanning tree dissimilarity* $d_{ST}(x, \overline{x})$.[14] In identical conditions, Ipsen and Mikhailov introduced in 2002 the *Ipsen–Mikhailov distance*

[9] See Equation 2.1 of [6].
[10] See Equation 2.2 of [6]
[11] See Section II-A of [6]
[12] See Equation 3.1 of [6].
[13] https://www.math.purdue.edu/liu1957/MA262_files/eigen.pdf
[14] See Equation 3.2 of [6].

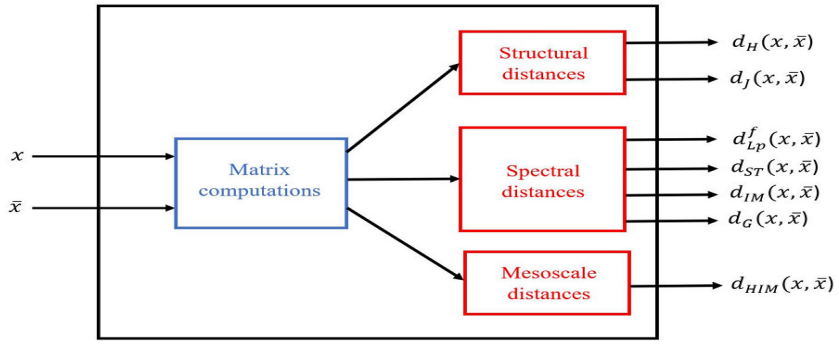


FIGURE 3. Graph comparison using MBTs.

$d_{IM}(x,\bar{x})$[15] that compares two graphs according to their spectral densities. In 2016, Gu et al. [24] rather proposed to consider the *continuous* ℓ_1 *distance* $d_G(x,\bar{x})$ between kernelized versions of the eigenvalue distributions using a *Gaussian kernel*.

3) MESOSCALE DISTANCES

In 2015, Jurman et al. [25] decided to combine the *Ipsen–Mikhailov distance* and the normalized *Hamming distance* by proposing the *Hamming–Ipsen–Mikhailov distance* $d_{HIM}(x,\bar{x})$.[16] Other mesoscale distances including the *connectivity-based distance* and the *Heat spectral wavelets distance* are also analyzed in [6].[17] Figure 3 depicts an overview of all the distances between two graphs using MBTs reviewed in this paper.

D. GRAPH KERNELS

1) DEFINITION OF GRAPH KERNELS

A *Hilbert space* is a vector space equipped with an inner product $\oplus$, which allows distances as well as angles to be measured, and orthogonality to be defined. As an example, for $m \geq 1$, the set $\mathbb{R}^m$ is a Hilbert space where $\oplus$ is the scalar product of $\mathbb{R}^m$.

Given a non-empty set $\mathbb{G}$ of graphs, a function $K : \mathbb{G}\times\mathbb{G} \rightarrow \mathbb{R}$ is a *graph kernel* if there is a Hilbert space $\mathcal{H}$ and a mapping function $\phi : \mathbb{G} \rightarrow \mathcal{H}$ that associates a feature vector with each graph such that Equation 1 is verified.

$$K(x,\bar{x}) = \phi(x) \oplus \phi(\bar{x}) \tag{1}$$

A graph kernel K is *explicit* if it explicitly characterizes the representations of the feature vectors $\phi(x)$ and $\phi(\bar{x})$ associated with the two graphs. Otherwise, K is *implicit* and produces directly the final result $K(x,\bar{x})$, the exact representations of the feature vectors being hidden.

2) COMPARISON USING GRAPH KERNELS

A graph kernel K is a positive semi-definite function (i.e: $K(x,x) \geq 0$ for all $x \in \mathbb{G}$) that can be directly used as a similarity measure between two graphs [1].[18] But in mathematics, distance measures (metrics) are preferable to similarity measures because the triangular inequality is not verified for similarity measures [26].[19]

When a kernel K is implicit, it is possible to compare a pair $(x,\bar{x}) \in \mathbb{G}\times\mathbb{G}$ of graphs using the *kernel metric* [14][20] derived from K defined in Equation 2.

$$\begin{aligned} d_{KI}(x,\bar{x}) &= ||\phi(x) - \phi(\bar{x})|| \\ &= \sqrt{K(x,x) + K(\bar{x},\bar{x}) - 2K(x,\bar{x})} \end{aligned} \tag{2}$$

When K is explicit, the graph comparison can further be performed through the comparison of the feature vectors associated with the two graphs using any valid vector distance $d_{\mathcal{H}}$ in the Hilbert space $\mathcal{H}$ as described in Equation 3. In this same context, the feature vectors generated by the mapping function ϕ can also serve as input data in a context of graph classification or graph clustering. Figure 4 summarizes the different measures for graph comparison using graph kernels reviewed in this paper.

$$d_{KE}(x,\bar{x}) = d_{\mathcal{H}}(\phi(x),\phi(\bar{x})) \tag{3}$$

3) MAJOR GRAPH KERNELS

Graph kernels have been widely used for comparing graphs since many decades and recent surveys related to graph kernels are proposed in [14] and [1]. It is not reasonable to realize an exhaustive review of the main existing graph kernels in the current paper for space constraints. A detailed chronological list of the major graph kernels proposed for classification purposes between years 1973 and 2019 is available in [14].[21] Additionally, [1] contains detailed computation schemes of several graph kernels (alongside with their time complexities), including: *vertex histogram*, *edge histogram*, *random walk*, *cyclic pattern*, *shortest path*, *Graphlet*, *Weisfeiler-Lehman subtree*, *neighborhood*

[15] See Equation 3.5 of [6].
[16] See Section 3.3.3 of [6].
[17] See Section IV of [6].
[18] See Section III-B of [1].
[19] See Definitions 1 et 2 of [26].
[20] See Equations 10 and 11 of [14].
[21] See Figure 2 of [14].

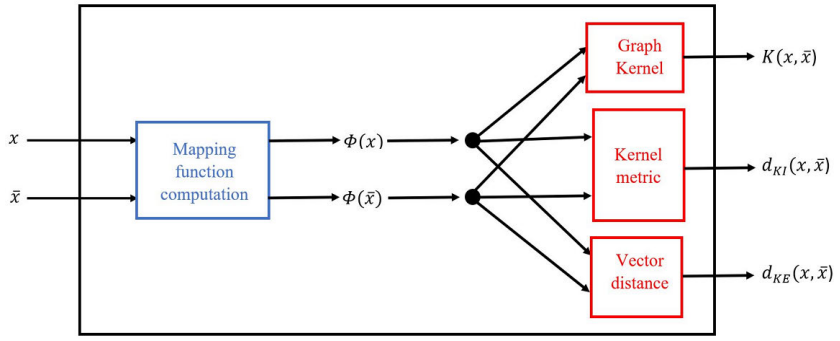


FIGURE 4. Graph comparison using GKs.

hash, *SVM-ϑ*, *pyramid match*, *Weisfeiler-Lehman optimal assignment*, *subgraph matching*, *GraphHopper*, *propagation*, etc.

4) PARADIGMS OF GRAPH KERNELS

A graph kernel that is capable of distinguishing between all non-isomorphic graphs in the feature space is a *complete graph kernel*. This completeness exclusively occurs when the corresponding mapping function ϕ is injective. But in 2003, Gärtner [27] demonstrated that computing a complete graph kernel is at least as hard as deciding whether two graphs are isomorphic and in 2005, Johnson demonstrated that there is no known polynomial-time algorithm for testing graph isomorphism [28]. Graph kernels are therefore designed according to one of the three following paradigms: *assignment*, *intersection* or *R-convolution*. Given two graphs x and $\bar{x}$:

1) *Assignment* graph kernels decompose x and $\bar{x}$ into substructures (i.e: vertices), then compute a matching between the substructures of x and those of $\bar{x}$ such that the overall similarity of the two graphs is maximized [1].[22]
2) *Intersection* graph kernels decompose x and $\bar{x}$ into diverse types of patterns (i.e: cyclic patterns or tree patterns) then count the number of common patterns of type that occur in the two graphs [1] [22].
3) *R-convolution* graph kernels decompose x and $\bar{x}$ into substructures (i.e: vertices or subgraphs), then evaluate a kernel between each pair $(u, \bar{u})$ of such structures with $u \in x$ and $\bar{u} \in \bar{x}$ [14].[23]

5) TAXONOMY OF GRAPH KERNELS

Existing graphs kernels can be organized into different categories depending on several criteria. Indeed, they can be divided into groups according to:

1) Their ability to handle unlabeled, vertex-labeled or vertex-attributed graphs.
2) The fact that they are explicit or implicit.
3) The design paradigm that they follow (*assignment*, *intersection* or *R-convolution*).

The taxonomy depicted in the Figure 3 of [1] emerges from the combination of all these criteria. This figure reveals that most of the existing graph kernels handle vertex-labeled graphs and are *R-convolution* graph kernels.

6) CHOOSING THE GOOD GRAPH KERNEL

The choice to prioritize or not a given graph kernel for a specific task is actually conditioned by the four following properties [14]:

1) The importance and nature of vertex attributes.
2) The size and density of graphs.
3) The importance of global graph structure.
4) The number of graphs in the dataset (for classification).

The Figure 10 of [14] presents guidelines for prioritizing graph kernels based on the four abovementionned properties. An important limitation of graph kernel arising from that figure is the fact that each graph kernel only prioritizes a predefined and limited set of graph properties. For instance, the *vertex histogram* kernel is adapted for comparing large graphs, but is not suitable when the graphs have continuous vertex attributes. On the contrary, the *subgraph matching* kernel is suitable for comparing graphs having continuous vertex attributes, but is not adapted for comparing large graphs. More generally, none of the graph kernels found in that Figure is adapted for comparing large graphs that have continuous vertex attributes.

E. GRAPH NEURAL NETWORKS

1) DEFINITION OF GNN

Assume that each vertex u of a graph is initially characterized by one multidimensional feature vector $\vec{u}$ whose components can for example include: its attributes, its degree, its label, etc. Given a graph $x \in \mathbb{G}$, a *GNN* can be viewed as a mapping function $\theta : \mathbb{G} \rightarrow \mathbb{R}^m$ that uses the structure of x as network support for deriving its associated descriptor vector $\theta(x)$ through an iterative learning process of its vertex feature vectors [15].[24] Hence, unlike a graph kernel whose mapping

[22] See Section 4.8 of [1].
[23] See Definition 1 of [14].
[24] See Section II of [15].

function ϕ results from a direct computation based on the graph substructures or on the graph patterns, the mapping function θ of a GNN is learnable.

Given a vertex $u \in V$, a GNN iteratively updates the components of its feature vector by first aggregating the feature vectors of its immediate neighboring vertices, then by combining the collected feature vectors with its actual feature vector. Proceeding this way, the GNN roughly simulates the functioning of a neural network composed of $\mathcal{K}$ layers such that at each iteration k, the structure of the graph x is considered as the k-th layer of the GNN, with $k = 1, \ldots, \mathcal{K}$ [2].[25]

2) DESIGN OF A SIMPLE GNN

Let $u^{(k)}$ be the feature vector characterizing the vertex u after k iterations of the learning process and let $\mathcal{N}(u) = \{v \in V \mid (u, v) \in E\}$ represents the set containing the immediate neighboring vertices of u. In these conditions, Equation 4 enables to update the components of $u^{(k)}$ with $k = 1, \ldots, \mathcal{K}$.

$$\begin{cases} u^{(k)} = \textbf{combine}\left(u^{(k-1)}, a^{(k)}\right) \text{ where} \\ a^{(k)} = \textbf{aggregate}(\{v^{(k-1)} \mid v \in \mathcal{N}(u)\}) \text{ and} \\ u^{(0)} = \vec{u}, \forall u \in V \end{cases} \tag{4}$$

The descriptor vector $\theta(x)$ associated with x is obtained from the $\mathcal{K}$-th layer through the aggregation of the feature vectors characterizing all the vertices of x as shown in Equation 5 where **readout** is the function performing the final aggregation. In these conditions, Figure 5 describes the computation scheme of $\theta(x)$ for a graph x where $V = \{u_1, \ldots, u_m\}$.

$$\theta(x) = \textbf{readout}\left(\{u^{(\mathcal{K})} \mid u \in V\}\right) \tag{5}$$

Several existing implementations of simple GNNs architectures proposed in 2017 [29], [30] and 2018 [31], [32], [33] rely on Equations 4 and 5.

3) DESIGN OF A POWERFUL GNN

It is possible to empower the overall learning efficiency of a simple GNN by inserting an additional user-selected *deep model* [34] inside each layer just after the graph structure. This leads to a pipeline embedding a dual learning process: the first being supported by the structure of the graph and the second by the structure of the deep model. One can decide to insert a unique deep model inside all the $\mathcal{K}$ layers, or to rather insert different deep models inside each layer [16].[26] According to this principle and in the conditions of Section II-E2, the components of the feature vector $u^{(k)}$ are now updated by Equation 6 with $k = 1, \ldots, \mathcal{K}$. In that equation, $\psi^{(k)}$ refers to the specific deep model inserted inside the k-th layer and that takes as input the vector resulting from the learning process supported by the graph structure.

$$\begin{cases} u^{(k)} = \psi^{(k)}(c^{(k)}) \text{ where} \\ c^{(k)} = \textbf{combine}\left(u^{(k-1)}, a^{(k)}\right) \text{ and} \\ a^{(k)} = \textbf{aggregate}(\{v^{(k-1)} \mid v \in \mathcal{N}(u)\}) \text{ and} \\ u^{(0)} = \vec{u}, \forall u \in V \end{cases} \tag{6}$$

Unlike a simple GNN where the mapping function only aggregates the feature vectors obtained from the $\mathcal{K}$-th layer, an empowered GNN has a mapping function that rather concatenates all the information aggregated across all the layers as shown in Equation 7. In these conditions, Figure 6 describes the computation scheme of $\theta(x)$ for a graph x where $V = \{u_1, \ldots, u_m\}$.

$$\theta(x) = \underbrace{\textbf{concat}}_{k=0,\ldots,\mathcal{K}}\left(\textbf{readout}\left(\{u^{(k)} \mid u \in V\}\right)\right) \tag{7}$$

Depending on the nature of deep models integrated into a powerful GNN, the following categories of GNNs have yet been proposed [16][27]: *Recurrent graph neural networks* (RecGNNs), *Convolutional Graph Neural Networks* (ConvGNNs), *Graph Autoencoders* (GAEs) and *Spatial-temporal Graph Neural Networks* (STGNNs). Technical details related to each of these categories are available in [16][28] and many large lists of relevant existing architectures based on these categories are available in that same work, alongside with their corresponding time complexities [16].[29] Open source implementations of these deep models are listed in [2].[30]

A simple example of such a powerful GNN that uses *multilayer perceptrons* (MLP) as deep models is offered by *graph isomorphic networks* (GIN)-ϵ whose general principle is described in Equation 8. In that equation, $\epsilon^{(k)}$ can be either be learnable, or a fixed user-defined scalar [15], [16].[31]

$$\begin{cases} \theta(x) = \underbrace{\textbf{concat}}_{k=0,\ldots,\mathcal{K}}\left(\sum_{u \in V} u^{(k)}\right) \text{ where} \\ u^{(k)} = \text{MLP}^{(k)}(c^{(k)}) \text{ and} \\ c^{(k)} = (1 + \epsilon^{(k)}).u^{(k-1)} + a^{(k)} \text{ and} \\ a^{(k)} = \sum_{v \in \mathcal{N}(u)} v^{(k-1)} \text{ and} \\ u^{(0)} = \vec{u}, \forall u \in V \end{cases} \tag{8}$$

4) COMPARISON USING GNNS

Given a GNN θ, the comparison between two graphs x and $\overline{x}$ is realized through the comparison of the feature vectors associated with the two graphs using any valid vector distance $d_{\mathcal{R}}$[32] (i.e: *Euclidean* distance, *Manhattan* distance, etc.) as described in Equation 9. Figure 7 summarizes this process.

$$d_\theta(x, \overline{x}) = d_{\mathcal{R}}(\theta(x), \theta(\overline{x})) \tag{9}$$

[25] See Figure 2 of [2].
[26] See Figure 3 of [16].
[27] See Figure 2 and Section III-A of [16].
[28] See Sections IV to VII of [16].
[29] See Tables 2 to 5 of [16].
[30] See Table B.7 of Appendix B in [2].
[31] See Equation 4.1 of [15] or Equation 23 of [16].
[32] See Equation 1 of [35].

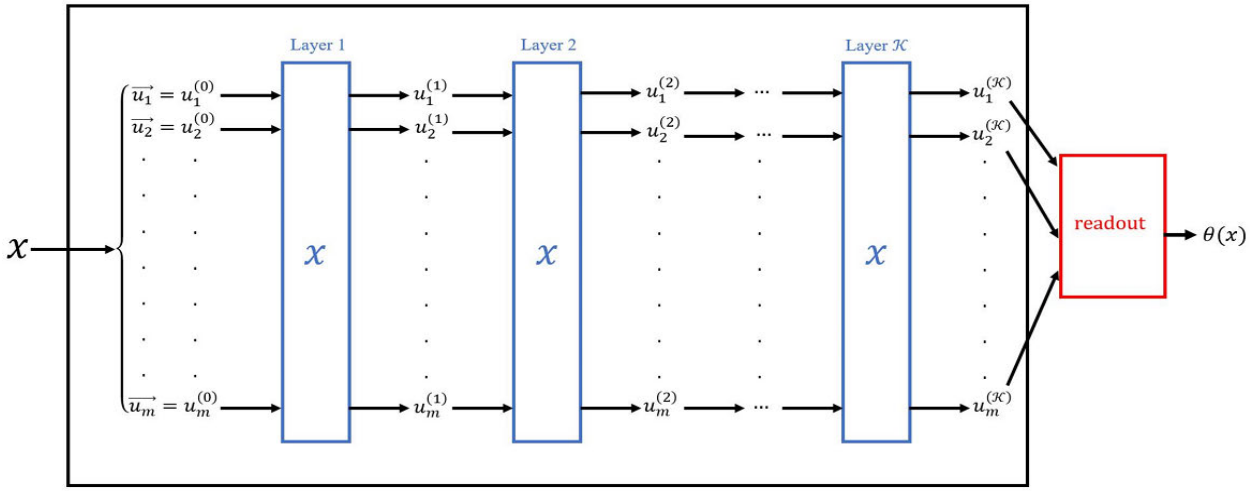


FIGURE 5. Computation of $\theta(x)$ for a graph x in a simple GNN.

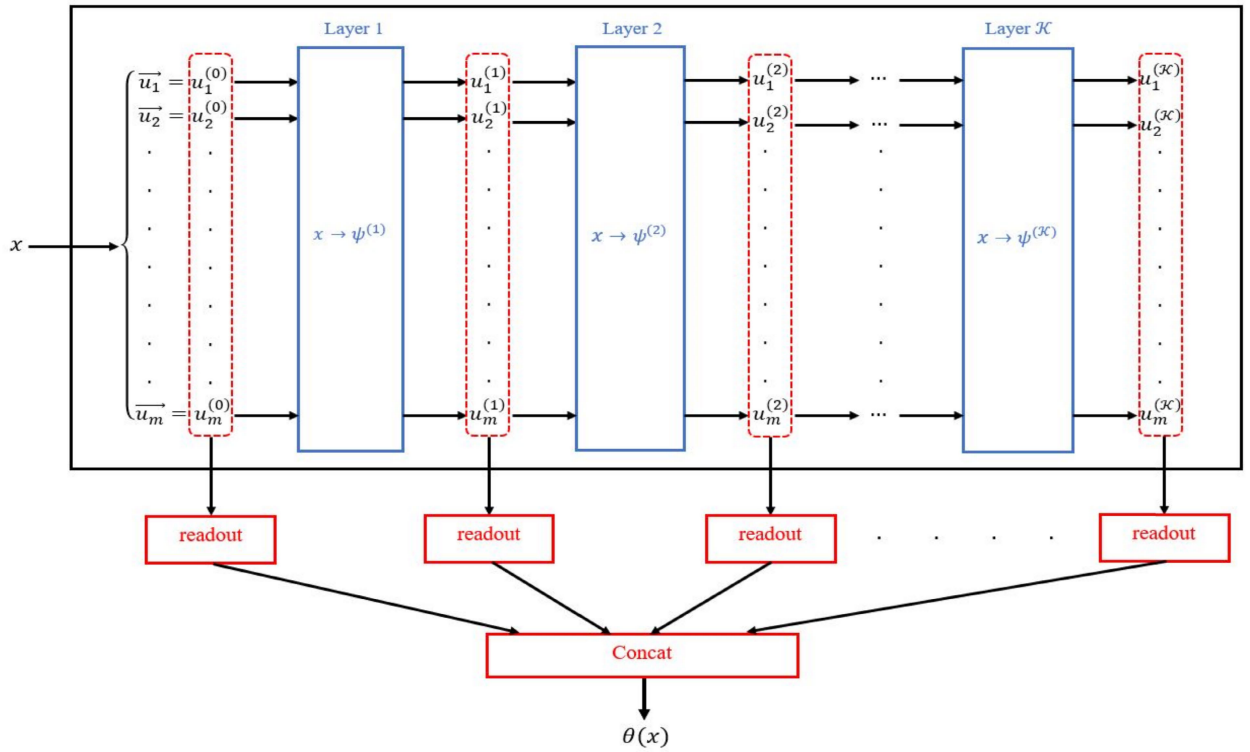


FIGURE 6. Computation of $\theta(x)$ for a graph x in a powerful GNN.

5) GNNS VS GKS

As it was the case for graph kernels, the feature vectors generated by θ can serve as inputs for graph classification or graph clustering. But, unlike graph kernels which prioritize predefined and limited sets of graph properties, GNNs can compare any two graphs, irrespective of their properties. Indeed:

1) The multidimensional feature vector $\vec{u}$ can embed various types of information related to each vertex u (i.e: attributes, degree, label, etc). GNNs can consequently handle labeled and attributed graphs (having continuous attributes).
2) The graph structure is used as network support, GNNs can therefore handle large graphs as well as dense graphs.
3) The functions **combine** and **aggregate** capture local changes around each vertex, while the function **readout** captures global changes across the graph. GNNs

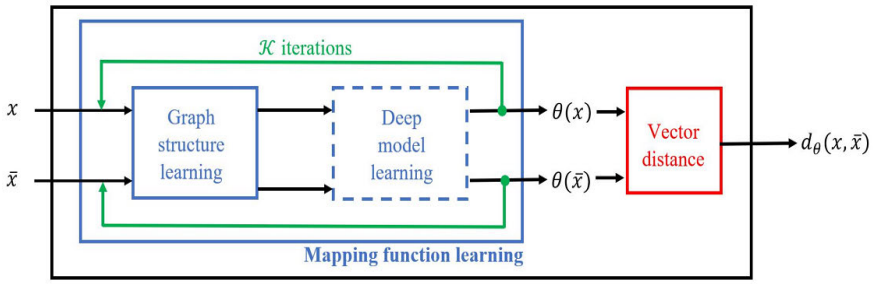


FIGURE 7. Graph comparison using GNNs.

are thus suitable for comparing graphs where the global changes are important, as well as graphs where local changes are important.

Unfortunately, GNNs suffer form the following general drawbacks:

1) They require lots of computing resources due to the integration of $\mathcal{K}$ deep models in the architecture that along the way, induce a high additional time cost required for the deep learning processes.
2) The components of the final feature vector $\theta(x)$ of a graph x are less interpretable and less explainable. Indeed, the explainability and interpretability of the results of deep models remain open scientific problems recently surveyed by Samek et al. in 2021 [36] and by Saleem et al. in 2022 [37].
3) The hyper-parameters of the deep models are difficult to adjust.

Table 2 summarizes the main information related to the existing techniques for graph comparison reviewed in this paper.

F. PROBLEM STATEMENT

Existing SSBTs for graph comparison [7], [8], [9], [10], [11], [12], [13] induce a high computation time. This limitation is attenuated by MBTs [6], which compare the graphs through operations performed on their matrices (adjacency, Laplacian), but they only enable the comparison of aligned graphs. GKs [1], [14] enable to overcome this drawback by embedding the graphs into vector spaces using a deterministic mapping function, but they only prioritize a predefined and limited set of graph properties. A good alternative for avoiding this limitation is offered by GNNs [2], [15], [16], which evaluate the mapping function through a deep learning process. Nevertheless, using GNNs requires lots of computing resources and the hyper-parameters of the user-selected deep models are difficult to adjust. Furthermore, the resulting final feature vectors associated with the input graphs are less interpretable/explainable. Beside this, existing techniques do not allow the explicit specification of user-defined properties on which the comparison should be performed. This analysis leads to the following research problem: *How to perform efficient graph comparison, based on an explicitly specified user-defined set of criteria?*

This paper attempts to overcome these limitations by proposing a customizable technique based on HMMs for comparing two graphs according to a user-defined set P of properties. HMMs are preferred here because they are suitable for learning sequential data and a vertex u can be viewed as the sequential list L_u of visited vertices produced by *DFS* during the traversal of the subgraph $\mathcal{G}(u)$ materializing a specific neighborhood of u. The fact that a HMM can be trained for learning multiple sequences offers the possibility of associating a unique HMM with a graph x. This model will learn the sequences derived from the $|V|$ subgraphs $\mathcal{G}(u)$, with $u \in V$.

III. PRESENTATION OF HMMS

A. DEFINITION OF A HMM

A HMM $\lambda = (A, B, \pi)$ is fully characterized by [38][33]:

1) Its number N of states, the set of states being $S = \{s_1, \ldots, s_N\}$. The state of the model at time t is generally noted $q_t \in S$.
2) Its number M of symbols, the set of symbols being $\vartheta = \{z_1, \ldots, z_M\}$. The symbol observed at time t is generally noted $o_t \in \vartheta$.
3) Its state transition probability matrix A verifying $A[s_i, s_j] = Prob(q_{t+1} = s_j | q_t = s_i)$ with $1 \leq i, j \leq N$.
4) Its symbols observation probability matrix B verifying $B[s_i, z_k] = Prob(z_k \text{ at time } t | q_t = s_i)$ with $1 \leq i \leq N$ and $1 \leq k \leq M$.
5) Its initial state probability vector π verifying $\pi[s_i] = Prob(q_1 = s_i)$ with $1 \leq i \leq N$.

B. GENERATION OF SEQUENCE

A HMM $\lambda = (A, B, \pi)$ can be used for generating a sequence $O = o_1 \ldots o_T$ composed of T symbols observed following the sequence of states $q = q_1 \ldots q_T$ as described in the *Markov chain* (MC) shown in Figure 8.

In order to obtain this MC, the following algorithm is executed:

1) Select the initial state s_j according to π and set $t = 0$.
2) Set $t = t + 1$ and change the current state to $q_t = s_j$

[33] See Section II-B of [38].

TABLE 2. Main existing techniques for comparing two graphs.

Main techniques	Surveyed by	Specific techniques	Specific drawbacks
Substructure based	-	Vertex-to-vertex [7, 12, 13] Edge-to-edge [8, 9] Subgraph-to-subgraph [10, 11]	High computation complexity
Matrix based	Donnat [6] (2018)	Structural distances Spectral distances [22, 23, 24] Mesoscale distances [25]	Only compare aligned graphs
Graph kernels	Kriege [14] (2020) Nikolentzos [1] (2021)	Assignment GKs Intersection GKs R-convolution GKs	Prioritize a limited set of graph properties
Graph neural networks	Xu et al. [15] (2018) Zhou [2] (2020) Wu [16] (2020)	Simple GNNs RecGNNs ConvGNNs GAEs STGNNs	-Lots of computing resources. -Less interpretable/explainable. -Parameters difficult to adjust.

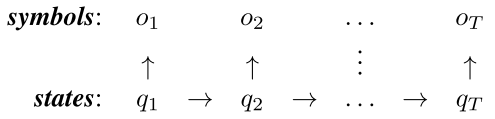


FIGURE 8. HMM used as sequence generator.

3) Select the symbol $o_t \in \vartheta$ to be observed at state q_t according to B.
4) If $(t < T)$ **go to** step 5, else **terminate**.
5) Select the state transition to be realized from the current state q_t to another state $s_j \in S$ according to A, then **go to** step 2.

C. MANIPULATION OF A HMM

Consider a sequence of symbols $O = o_1 \ldots o_T$ and a HMM $\lambda = (A, B, \pi)$. The probability $Prob(O|\lambda)$ to observe O given λ is efficiently calculated by the *Forward-Backward* algorithm [38][34] that runs in $\mathcal{O}(T.N^2)$. Given a sequence of symbols $O = o_1 \ldots o_T$, it is possible to iteratively re-estimate the parameters of a HMM $\lambda = (A, B, \pi)$ in order to maximize the value of $Prob(O|\overline{\lambda})$, where $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi})$ is the re-estimated model. The *Baum-Welch* algorithm [38][35] is used for this purpose. This algorithm runs in $\mathcal{O}(\gamma.T.N^2)$ where γ is the user-defined maximum number of iterations.

The *Baum-Welch* algorithm can also be used to train a HMM for learning multiple sequences.[36] In this case, the algorithm maximizes the value of $Prob(O|\overline{\lambda}) = \sum_{k=1}^{K} Prob(O^{(k)}|\overline{\lambda})$ where $O = \{O^{(1)}, \ldots, O^{(K)}\}$ is a set of K sequences and $O^{(k)} = o_1^{(k)} \ldots o_{T_k}^{(k)}$ is the k^{th} sequence of O. The time cost of the *Baum-Welch* algorithm for multiple sequences is approximated by $\mathcal{O}\left(\gamma.(\sum_{k=1}^{K} T_k).N^2\right)$

[34] See Section III-A of [38].
[35] See Section III-C of [38].
[36] See Section V-B of [38].

D. STATIONARY DISTRIBUTION

A vector $\varphi = (\varphi[s_1], \ldots, \varphi[s_N])$ is a stationary distribution of a HMM $\lambda = (A, B, \pi)$ if [39][37]:

1) $\forall j, \varphi[s_j] \geq 0$ and $\sum_j \varphi[s_j] = 1$
2) $\varphi = \varphi.A \Leftrightarrow \left(\varphi[s_j] = \sum_i \varphi[s_i] \times A[s_i, s_j], \forall j\right)$

$\varphi[s_j]$ estimates the overall proportion of time spent by λ in state s_j after a sufficiently long time. φ can be extracted from any line of the matrix $A^r = A \times A \times \ldots \times A$ (r times) when $r \to +\infty$. The computation of φ requires $\mathcal{O}(r.N^3)$ arithmetic operations.

IV. THE PROPOSED APPROACH

A. MAIN IDEA

Consider a graph x and a user-defined property p which is a function either applicable to any vertex or to any edge of x. Our intuition is that x can be accurately characterized by capturing the adherence of the neighborhood of each vertex $u \in V$ to the property p. To achieve this goal, for every vertex $u \in V$ and given a user-defined constant β, we first define $\mathcal{G}(u)$ as the subgraph of x explored by DFS during the traversal of x starting from u and such that the value of the variable *depth* is always less or equal to β.

Algorithm 3 DFS_new

Inputs: $x = (V, E)$, $u \in V$, β, p
Output: L_u, $\tilde{L}_u$

1: $L_u \leftarrow \emptyset$
2: $\tilde{L}_u \leftarrow \emptyset$
3: $depth \leftarrow 0$
4: **observe_vertex**($depth, p(u)$)
5: DFSrec_new($x, L_u, \tilde{L}_u, u, \beta, depth, p$)
6: **return** $L_u, \tilde{L}_u$

Algorithm 3 (which calls Algorithm 4) is a modified version of DFS executed on the graph x starting from u, but that limits the traversal to the subgraph $\mathcal{G}(u)$. In this modified

[37] See Definition 9.1 of [39].

Algorithm 4 DFSrec_new

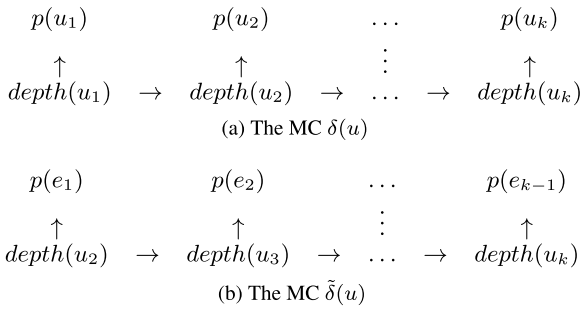
Inputs: $x = (V, E), L, \tilde{L}, u \in V, \beta, depth, p$
Output: $L, \tilde{L}$

1: Insert(u, L)
2: $Z \leftarrow \{v \in V | (u, v) \in E\}$
3: **for each** ($v \in Z$) **do**
4: **if** ($v \notin L$) **then**
5: $depth \leftarrow depth + 1$
6: **if** ($depth \leq \beta$) **then**
7: $e \leftarrow (u, v)$
8: Insert($e, \tilde{L}$)
9: **observe_vertex**($depth, p(v)$)
10: **observe_edge**($depth, p(e)$)
11: DFSrec_new($x, L, \tilde{L}, v, \beta, depth, p$)
12: **end if**
13: $depth \leftarrow depth - 1$
14: **end if**
15: **end for**
16: **return** $L, \tilde{L}$

version, the function **PreUse** is replaced by the functions **observe_vertex** and **observe_edge** that observe the value of the property p at the current *depth*, respectively for currently visited vertex and the currently visited edge. The function **PostUse** is removed in these algorithms. In practice, if p is designed for vertices, then the line 10 of Algorithm 4 must be ignored. But if p is on the contrary designed for edges, the line 4 of Algorithm 3 and the line 9 of Algorithm 4 must be ignored in this case.

The main idea of this work arises from the following observation we made about the traversal of $\mathcal{G}(u)$ using Algorithm 3. When *DFS* traverses $\mathcal{G}(u)$ starting from u, the algorithm sequentially transits from one visited vertex located at a specific *depth*, to another visited vertex located at another *depth*, while observing the current value of property p at each step, until the last vertex is visited. In other words, *DFS sequentially transits from one depth to the other and at each step, the algorithm observes the value of the property for the currently visited vertex/edge*. This analysis enables us to transform $\mathcal{G}(u)$ into a generated MC resulting from the traversal of $\mathcal{G}(u)$ using the modified DFS starting from u. In this transformation, the hidden states are the depths that belong to $\{0, 1, \ldots, \beta\}$ and the observed symbols are the values of the property p.

For any vertex u, assume that $depth(u)$ refers to the value of the variable *depth* taken with u as input parameters of Algorithm 4. In these conditions, if $L_u = [u_1, u_2, \ldots, u_k]$ and $\tilde{L}_u = [e_1, e_2, \ldots, e_{k-1}]$ are respectively the ordered lists of visited vertices and visited edges returned by Algorithm 4 (i.e: $u_1 = u$ and $depth(u_1) = 0$), then Figures 9a and 9b show the resulting MCs $\delta(u)$ and $\tilde{\delta}(u)$ associated with the subgraph $\mathcal{G}(u)$. Depending on the target (vertex or edge) of the property p, only one of these two MCs is obtained after the traversal of $\mathcal{G}(u)$.



(a) The MC $\delta(u)$

(b) The MC $\tilde{\delta}(u)$

FIGURE 9. MCs derived from the traversal of $\mathcal{G}(u)$.

For each vertex u of the graph x, the traversal of the neighborhood $\mathcal{G}(u)$ of u can be performed for deriving one MC associated with u as described in Figures 9a and 9b. All the resulting distinct MCs can later serve as inputs for initializing and then for training a unique HMM associated with the graph x using the *Baum-Welch* algorithm for multiple sequences. This model will learn the adherence of the graph x to the property p through the overall analysis of the subgraphs $\mathcal{G}(u)$, for every vertex u of x. The authors of [40][38] proceeded analogically to transform a tree into a MC given a user-defined property with the final goal of comparing two finite tree sets using HMMs.

This modeling principle can be applied using not only one single property p, but a finite set $P = \{p_1, \ldots, p_m\}$ of user-defined properties. In these conditions, one HMM $\lambda_i(x)$ is associated with the graph x, for each property $p_i \in P$, with $(1 \leq i \leq m)$. As it was already explained in [35][39] where the authors compared finite sets of vectors using HMMs, it is not systematically necessary to determine the most relevant and informative content of P since the result of the comparison always remains consistent according to P, whatever it contains. Indeed, it is only the final goal of the comparison that indicates the necessity of determining the best content of P. For pure decision support as comparing two graphs in order to select the graph that matches the best with specific user-defined preferences, the content of P is fully determined by the user preferences. But, the discovery of the best content of P can become necessary and challenging for classification purposes if the final goal is to obtain the best classification performances. However, it is also possible to conduct classification experiments with the aim of evaluating to what extent the properties in P are suitable for distinguishing the classes found in the experimental dataset.

Each property in P can target the graph structure, the vertex/edge labels or the vertex/edge attributes. In order to show the flexibility of the proposed approach regarding the choice of the properties in P, consider a graph x and a vertex $u \in V$, both taken as input by Algorithm 4. If v is the currently visited vertex in Algorithm 4, then the arbitrary properties p_1 to p_{12} listed in Table 3 can for example belong to P.

[38] See Section IV-A of [40].
[39] See Section V-B of [35].

TABLE 3. Examples of arbitrary properties that can belong to P when u is the input vertex of Algorithm 4 and v is the currently visited vertex of this same algorithm.

Target	Related to	Property	Attempts to capture
Structure	The exact value of $deg(u)$	$p_1(u) = deg(u)$	The degree of u
		$p_2(u) = 100\times\frac{\lvert W_1\rvert}{deg(u)}$	The percentage of immediate neighbors of u having a degree lower to $deg(u)$
	The parity of $deg(u)$	$p_3(u) = deg(u)\%2$	The parity of $deg(u)$
		$p_4(u) = 100\times\frac{\lvert W_2\rvert}{deg(u)}$	The percentage of immediate neighbors of u whose degree has the same parity as the parity of $deg(u)$
	The last digit of $deg(u)$	$p_5(u) = deg(u)\%10$	The last digit of $deg(u)$
		$p_6(u) = 100\times\frac{\lvert W_3\rvert}{deg(u)}$	The percentage of immediate neighbors of u whose degree has a last digit lower to the last digit of $deg(u)$
Labels	The exact value of $l(u)$	$p_7(u) = l(u)$	The label of u
		$p_8(u) = 100\times\frac{\lvert W_4\rvert}{deg(u)}$	The percentage of immediate neighbors of u having the same label as the label of u
	The exact value of $l(u,v)$	$p_9(u,v) = l(u,v)$	The label of the edge (u,v)
		$p_{10}(u,v) = 100\times\frac{\lvert W_5\cup W_6\rvert}{deg(u)+deg(v)}$	The percentage of edges connected to (u,v) having the same label as the label of (u,v)
Attributes	The exact value of $f(u)$	$p_{11}(u) = f(u)$	The attribute of u
		$p_{12}(u) = 100\times\frac{\lvert W_7\rvert}{deg(u)}$	The percentage of immediate neighbors of u having an attribute lower to $f(u)$

The variables W_1 to W_7 used in that table are defined in Equation 10.

$$\begin{aligned}
W_1 &= \{w \mid (u,w)\in E \text{ and } deg(w) \leq deg(u)\}\\
W_2 &= \{w \mid (u,w)\in E \text{ and } (deg(w)\%2) = (deg(u)\%2)\}\\
W_3 &= \{w \mid (u,w)\in E \text{ and } (deg(w)\%10) \leq (deg(u)\%10)\}\\
W_4 &= \{w \mid (u,w)\in E \text{ and } l(w) = l(u)\}\\
W_5 &= \{w \mid (u,w)\in E \text{ and } l(u,w) = l(u,v)\}\\
W_6 &= \{w \mid (v,w)\in E \text{ and } l(v,w) = l(u,v)\}\\
W_7 &= \{w \mid (u,w)\in E \text{ and } f(w) \leq f(u)\} \qquad (10)
\end{aligned}$$

B. METHODOLOGY

Figure 10 depicts the methodology proposed in the current work for comparing two graphs x and $\overline{x}$ according to the set $P = \{p_1, \ldots, p_m\}$ of properties.

Given a user-defined positive integer β, this methodology is composed of the three following main steps:

1) **Graph learning**: For each property $p_i\in P$ with $(1\leq i\leq m)$, a HMM $\lambda_i(x)$ (resp. $\lambda_i(\overline{x})$) is initialized and trained for learning the adherence of the graph x (resp. $\overline{x}$) to the property p_i, through the overall analysis of the neighborhoods of its vertices.
2) **Vector computation**: Meta-data extracted from the resulting HMMs are used for generating one feature vector $\mu_i(x)$ associated with the graph x according to the property p_i. These meta-data are related to the overall time spent by the model observing each symbol after a sufficiently long time, irrespective of the state from which this observation is realized. The m resulting vectors are later concatenated for deriving a unique feature vector $\mu_P(x)$ associated with the graph x according to P. The same process is executed for generating a unique feature vector $\mu_P(\overline{x})$ associated with the graph $\overline{x}$ according to P.
3) **Vector comparison**: The comparison between the graphs x and $\overline{x}$ is finally performed through the comparison of their respective feature vectors $\mu_P(x)$ and $\mu_P(\overline{x})$ using any existing vector distance d.

C. GRAPH LEARNING

Consider a graph x, a positive integer β and a set $P = \{p_1, \ldots, p_m\}$ of properties. For each property $p_i\in P$ with $(1\leq i\leq m)$, the model $\lambda_i(x)$ associated with x according to the property p_i is derived following the methodology depicted in Figure 11 when $V = \{u_1, \ldots, u_k\}$ such that $(k = |V|)$. This figure is composed of the following three main steps: *Neighborhood exploration*, *Transformation into Markov chain* and *HMM initialization and training*.

1) NEIGHBORHOOD EXPLORATION

Given a user-defined integer β and for each vertex u_j of the graph x with $(1\leq j\leq k)$, Algorithm 3 is executed here to traverse the subgraph $\mathcal{G}(u_j)$. This algorithm will thus explore a specific neighborhood of the vertex u_j whose depth is proportional to the value of β. When $(\beta = 0)$, the algorithm only explores u_j. When $(\beta = 1)$, u_j and its the immediate neighbors are explored. Now when $(\beta = \beta_0)$ such that $(\beta_0\geq 2)$, the algorithm explores all the vertices that must be explored when $(\beta = \beta_0 - 1)$ and their immediate neighbors that have not yet been explored.

2) TRANSFORMATION INTO MARKOV CHAIN

For each vertex u_j of the graph x with $(1\leq j\leq k)$, the subgraph $\mathcal{G}(u_j)$ is transformed here into the MC $\delta_i(u_j)$ as described in Figure 9a (If p_i is designed for edges, $\tilde{\delta}_i(u_j)$ of Figure 9b is rather used). The resulting MCs are gathered into the set $\Delta_i = \{\delta_i(u_1), \ldots, \delta_i(u_k)\}$. Thereafter, duplicates are removed from Δ_i.

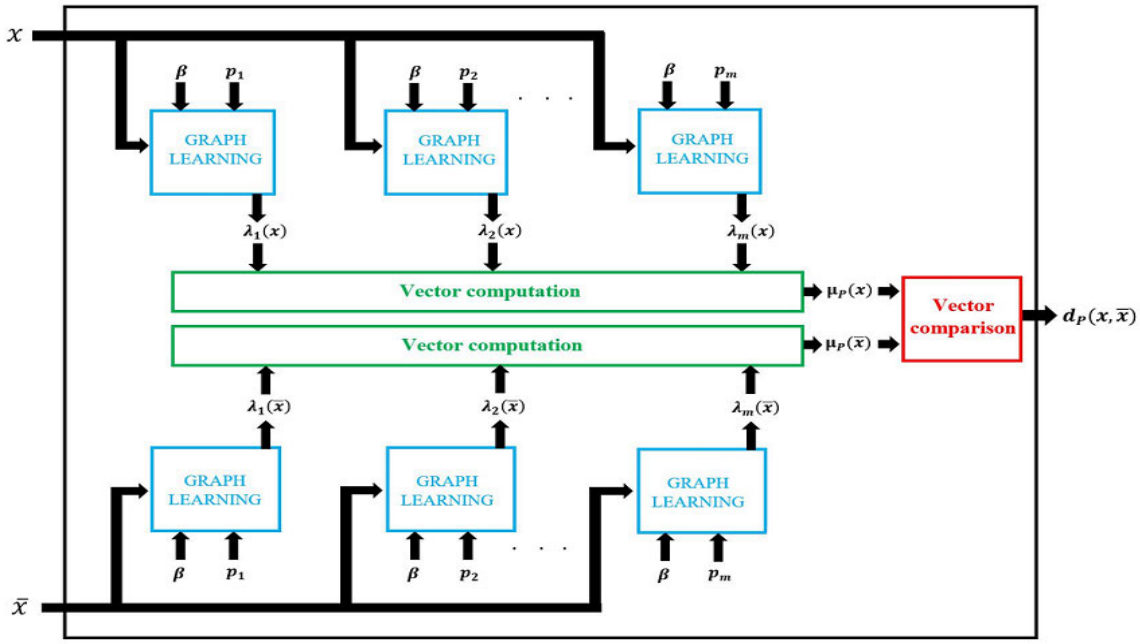


FIGURE 10. Methodology proposed for comparing two graphs x and $\bar{x}$ according to the set $P = \{p_1, \ldots, p_m\}$ of properties.

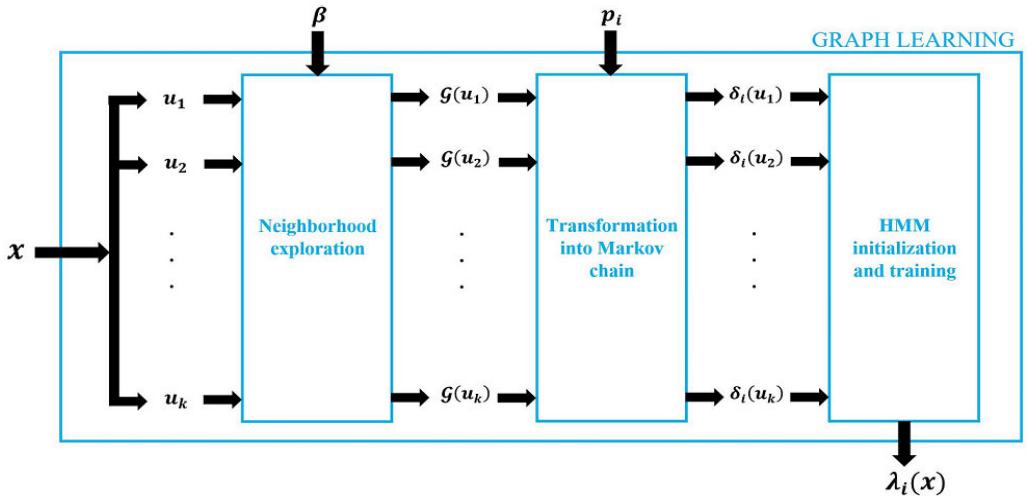


FIGURE 11. Methodology proposed for deriving the model $\lambda_i(x)$ associated with the graph x according to the property p_i.

TABLE 4. Example of sampling of the values of an integer $i \in \mathbb{N}$.

i	$[0,50]$	$[51,60]$	...	$[91,100]$	$[101,200]$	...	$[901,1000]$	$[1001,2000]$	...	$[4001,5000]$	> 5000
$\eta(i)$	i	51	...	56	57	...	65	66	...	69	70

Consider for example the vertex-labeled graph x depicted in Figure 2c. When Algorithm 3 is executed with $\beta = 1$ and $\beta = 2$ on the graph x, starting from the vertex u whose identifier is 4, the MCs shown in Figures 12a to 12h are derived for the properties p_1, p_3, p_7 and p_8 described in Table 3.

The main issue arising from the MCs presented in Figures 9a and 9b is that the symbols are the values of the property p, that can belong to any set I (finite or not) depending on the user preferences. This is a problem because in a HMM, the symbols must always belong to a finite set. But, as it can be observed in Table 3:

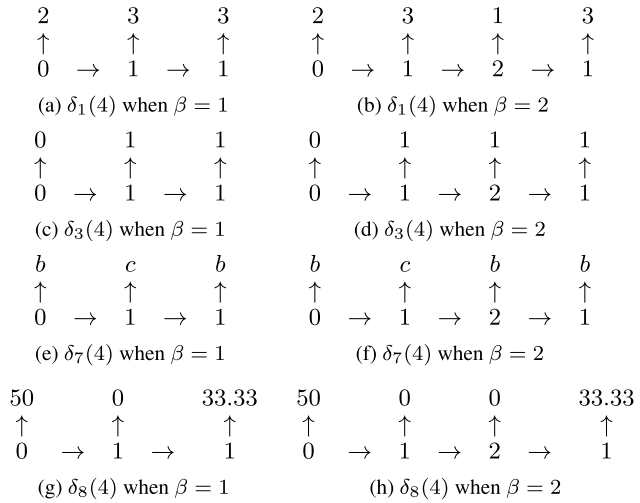


FIGURE 12. MCs associated with the vertex 4 of the graph x of Figure 2c for p_1, p_3, p_7 and p_8 when $\beta = 1$ and $\beta = 2$.

- Only p_3, p_5, p_7 and p_9 return values that belong to finite sets.
- p_1 returns a positive integer.
- p_{11} returns a real number.
- $p_2, p_4, p_6, p_8, p_{10}$ and p_{12} return percentages that belong to continuous interval [0, 100].

An obvious solution to this problem consists in sampling the set I into a finite number of samples and to consider each sample as a symbol. For properties like p_1 that return elements of $\mathbb{N}$, one can for example consider the sampling described in Table 4, which matches any integer $i\in\mathbb{N}$ with the sample $\eta(i)\in\{0, 1, \ldots, 70\}$. This sampling can of course be modified by the user, but in its actual state, it can accurately capture the exact value of the degree of a vertex, except for graphs where vertices have huge degrees. A suitable symmetry with respect to zero can be applied on Table 4 for matching any integer $i\in\mathbb{Z}$ with the sample $\eta(i)\in\{-70, \ldots, -1, 0, 1, \ldots, 70\}$. The MCs related to p_1 shown in Figures 12a and 12b remain unchanged after the application of this sampling.

For the case of properties that return percentages belonging to the continuous interval $I = [0, 100]$, one can adopt the sampling proposed in Equation 11 which matches any real number $i\in[0, 100]$ with the sample $\eta(i)\in\{0, 1, \ldots, M\}$ following [35],[40] where M is a user-defined integer. When this sampling is applied with $M = 10$ to the MCs related to p_8 initially presented in Figures 12g and 12h, the modified MCs shown in Figures 13a and 13b are obtained.

$$\begin{cases} (i = 0) \Rightarrow (\eta(i) = 0) \\ i\in\left]\dfrac{100}{M}\times(k-1), \dfrac{100}{M}\times k\right], (1\leq k\leq M) \Rightarrow (\eta(i) = k) \end{cases} \tag{11}$$

Given any vertex/edge j of a graph x, when the property p returns a real number that can be positive or negative,

[40] See Equation 3 of [35].

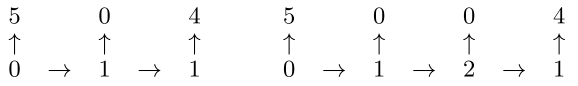


FIGURE 13. MCs associated with the vertex 4 of the graph x of Figure 2c for p_8 when Equation 11 is used with $M = 10$.

one can initially normalize this returned value according to the maximum value $maxi$ of p observed in the graph x as described in Equation 12 where k is any vertex/edge of x. This normalization will replace the original returned value $p(j)$ by the new value i that belongs to the continuous interval $[-100, 100]$. Equation 13 can then be used for matching any $i\in[-100, 100]$ with the sample $\eta(i)\in\{0, 1, \ldots, 2M\}$ following [35],[41] where M is a user-defined integer.

$$\begin{cases} i = 100\times\dfrac{p(j)}{maxi} \text{ where} \\ maxi = \max\{p(k) \mid k \text{ in } x\} \end{cases} \tag{12}$$

$$\begin{cases} i\in\left[-100\times\dfrac{k}{M}, -100\times\dfrac{(k-1)}{M}\right[, (1\leq k\leq M) \Rightarrow (\eta \\ (i) = k + M)(i = 0) \Rightarrow (\eta(i) = 0) \\ i\in\left]100\times\dfrac{(k-1)}{M}, 100\times\dfrac{k}{M}\right], (1\leq k\leq M) \Rightarrow (\eta(i) = k) \end{cases} \tag{13}$$

3) HMM INITIALIZATION AND TRAINING

Consider a graph x where $V = \{u_1, \ldots, u_k\}$ and a property $p_i\in P$ that returns values belonging to a set I, with $(1\leq i\leq m)$. Given two positive user-defined constants β and M we propose to parameter the initial HMM $\hat{\lambda}_i(x) = (\hat{A}_i(x), \hat{B}_i(x), \hat{\pi}_i(x))$ associated with x according to p_i as described below for statistically capturing the state transitions and the symbol probability distributions from the content of the set of MCs $\Delta_i = \{\delta_i(u_1), \ldots, \delta_i(u_k)\}$:

1) The set of states is $S = \{0, \ldots, \beta\}$.
2) The content of the set ϑ of symbols depends on the nature of the set I. If I is finite, then $(\vartheta = I)$. When $I = \mathbb{N}$, the sampling proposed in Table 4 can be used such that we have $\vartheta = \{0, 1, \ldots, 70\}$. One can perform a suitable symmetry with respect to zero of this same table in order to obtain $\vartheta = \{-70, \ldots, -1, 0, 1, \ldots, 70\}$ when $I = \mathbb{Z}$. For the case where I is the continuous interval $[0, 100]$, Equation 11 enables to sample this interval such that we obtain $\vartheta = \{0, 1, \ldots, M\}$. Finally, when $I = \mathbb{R}$, Equation 13 enables to obtain $\vartheta = \{0, 1, \ldots, 2M\}$ after normalizing the values of p_i using Equation 12.
3) The probability of transiting from state s_j to state s_l is calculated in Equation 14 where $transit(s_j, s_l, \Delta_i)$ is the number of transitions from state s_j to state s_l in Δ_i and $transit(s_j, -, \Delta_i)$ is the number of transitions from state

[41] See Equations 4 and 5 of [35].

s_j to any destination in Δ_i.

$$\hat{A}_i(x)[s_j, s_l] = \frac{transit(s_j, s_l, \Delta_i)}{transit(s_j, -, \Delta_i) + 1} \tag{14}$$

4) The probability to observe symbol z_l at state s_j is calculated in Equation 15 where $observe(z_l, s_j, \Delta_i)$ is the number of times symbol z_l is observed from state s_j in Δ_i, and $observe(-, s_j, \Delta_i)$ is the number of occurrences of state s_j in Δ_i.

$$\hat{B}_i(x)[s_j, z_l] = \frac{observe(z_l, s_j, \Delta_i)}{observe(-, s_j, \Delta_i) + 1} \tag{15}$$

5) The probability of starting with state s_j is calculated in Equation 16, where $start(s_j, \Delta_i)$ is the number of MCs in Δ_i starting with state s_j.

$$\hat{\pi}_i(x)[s_j] = \frac{start(s_j, \Delta_i)}{|\Delta_i| + 1} \tag{16}$$

The parameters of $\hat{\lambda}_i(x)$ are not probability distributions due to the digit '1' intentionally added to the denominators of its various components for avoiding eventual divisions by zero or zero-probabilities in the initial model. In order to solve this problem, an equitable redistribution of the missing quantity is applied to each element of each line in $\hat{\lambda}_i(x)$ to derive the readjusted initial HMM $\tilde{\lambda}_i(x) = (\tilde{A}_i(x), \tilde{B}_i(x), \tilde{\pi}_i(x))$ whose parameters are calculated in Equation 17.

$$\begin{cases} \tilde{A}_i(x)[s_j, s_l] = \hat{A}_i(x)[s_j, s_l] + \dfrac{1}{\beta + 1}\left(1 - \displaystyle\sum_{h=0}^{\beta} \hat{A}_i(x)[s_j, s_h]\right) \\ \tilde{B}_i(x)[s_j, z_l] = \hat{B}_i(x)[s_j, z_l] + \dfrac{1}{|\vartheta|}\left(1 - \displaystyle\sum_{h=0}^{|\vartheta|-1} \hat{B}_i(x)[s_j, z_h]\right) \\ \tilde{\pi}_i(x)[s_j] = \hat{\pi}_i(x)[s_j] + \dfrac{1}{\beta + 1}\left(1 - \displaystyle\sum_{h=0}^{\beta} \hat{\pi}_i(x)[s_h]\right) \end{cases} \tag{17}$$

Finally, the readjusted initial HMM $\tilde{\lambda}_i(x)$ is trained using the *Baum-Welch* algorithm for multiple sequences. This enables to derive the final HMM $\lambda_i(x)$ associated with the graph x according to the property p_i. The training sequences taken as inputs by the *Baum-Welch* algorithm are the sequences of symbols appearing in Δ_i. When this principle is executed for each property in P, the set $\{\lambda_1(x), \ldots, \lambda_m(x)\}$ of HMMs is obtained.

D. VECTOR COMPUTATION

In this section, meta-data are extracted from the HMM $\lambda_i(x)$ in order to generate one feature vector $\mu_i(x)$ associated with the graph x according to the property $p_i \in P$. The l^{th} component $\mu_i^l(x)$ of this vector is considered here as the overall proportion of time spent by $\lambda_i(x)$ observing the symbol z_l after a sufficiently long time, irrespective of the state from which this observation is realized. In order to compute the value of $\mu_i^l(x)$, one must first use the following principle for evaluating the overall proportion of time spent by $\lambda_i(x)$ observing the symbol z_l from state s_j after a sufficiently long time:

1) The overall proportion of time spent by $\lambda_i(x)$ in state s_j after a sufficiently long time is first evaluated. This proportion is given by the j^{th} component $\varphi_i(x)[s_j]$ of the stationary distribution of $\lambda_i(x)$.
2) The result obtained at step 1 is then multiplied by the probability of observing the symbol z_l from state s_j which is given by $B_i(x)[s_j, z_l]$.

The value of $\mu_i^l(x)$ is finally obtained by repeating this process for every state s_j and by summing the resulting proportions. Equation 18 summarizes the computation scheme of $\mu_i(x)$.

$$\begin{cases} \mu_i(x) = [\mu_i^1(x), \mu_i^2(x), \ldots, \mu_i^{|\vartheta|}(x)] \quad \text{where} \\ \mu_i^l(x) = \displaystyle\sum_{j=0}^{\beta} \left(\varphi_i(x)[s_j] \times B_i(x)[s_j, z_l]\right) \quad \text{with } (1 \leq l \leq |\vartheta|) \end{cases} \tag{18}$$

If p_i returns a real number, the value *maxi* used in Equation 12 for normalizing the values of p_i is inserted as the last additional component of $\mu_i(x)$ to take into account the effects of this normalization as shown in Equation 19.

$$\begin{cases} \mu_i(x) = [\mu_i^1(x), \mu_i^2(x), \ldots, \mu_i^{|\vartheta|}(x), maxi] \quad \text{where} \\ \mu_i^l(x) = \displaystyle\sum_{j=0}^{\beta} \left(\varphi_i(x)[s_j] \times B_i(x)[s_j, z_l]\right) \quad \text{with } (1 \leq l \leq |\vartheta|) \end{cases} \tag{19}$$

The final feature vector $\mu_P(x)$ associated with the graph x according to the set $P = \{p_1, \ldots, p_m\}$ is obtained by concatenating the components of the feature vectors generated for the m properties in P as described in Equation 20.

$$\mu_P(x) = \underbrace{\mathbf{concat}}_{i=1,\ldots,m} (\mu_i(x)) \tag{20}$$

E. VECTOR COMPARISON

Consider two graphs x and $\overline{x}$ that can be labeled or attributed. Given a user-defined set $P = \{p_1, \ldots, p_m\}$ of properties, the distance/similarity $d_P(x, \overline{x})$ between x and $\overline{x}$ is evaluated by comparing their respective associated final feature vectors using any valid distance/similarity $d_{\mathcal{R}}$ between vectors in $\mathbb{R}^{32}$ (*Euclidean*, *Manhattan*, etc.) as shown in Equation 21.

$$d_P(x, \overline{x}) = d_{\mathcal{R}}(\mu_P(x), \mu_P(\overline{x})) \tag{21}$$

F. TIME COST OF THE PROPOSED APPROACH

According to the methodology shown in Figure 10, the theoretical time cost of the proposed approach is dominated by the time required for all graph learning steps, augmented by the time required for computing the final feature vectors.

Consider a graph x and a set $P = \{p_1, \ldots, p_m\}$ of properties. If ε and p_{max} respectively designate the maximum vertex degree in x (i.e: $\varepsilon = \max\{deg(u) \mid u \in V\}$) and the maximum time required for computing the value of any

property in P, then the time cost of the graph learning for each property p_i can be approximated as follows according to the main steps described in Figure 11:

1) *Neighborhood exploration*: For each vertex $u \in V$, the traversal of $\mathcal{G}(u)$ visits $\varepsilon^0 = 1$ vertex at the depth 0 (this vertex is u), and at most ε^1 vertices at the depth 1 (these are the immediate neighbors of u), and at most ε^2 vertices at the depth 2 (these are the immediate neighbors of the immediate neighbors of u), and so on until the depth β where ε^β vertices are visited at most. Finally, $\left(\sum_{i=0}^{\beta} \varepsilon^i\right) = \left(\frac{1-\varepsilon^{(\beta+1)}}{1-\varepsilon}\right)$ vertices are at most visited during the traversal of $\mathcal{G}(u)$. Given that β is generally a small number, in addition with the fact that the effective degrees of the various vertices in the graph are most often lower than ε, the effective number of visited vertices is in practice very low compared to $|V|$. Nevertheless, the number of visited vertices can sometimes be equal to $|V|$ for small graphs or for high values of β. For this reason, we assume here that in the worst case, $|V|$ vertices are visited during the the traversal of $\mathcal{G}(u)$ for each vertex u.
2) *Transformation into MC*: Given that the property p_i is evaluated each time a new vertex is visited during the traversal of $\mathcal{G}(u)$ for each vertex $u \in V$, the time cost required for transforming all the vertices of x into MCs is $\left(p_{max}.|V|^2\right)$.
3) *HMM initialization and training*: The time required for initializing a HMM was experimentally very low. This step is consequently dominated by the model training phase, which according to Section III-C, runs in around $\gamma.(\sum_{k=1}^{|V|} |V|).(\beta+1)^2 = \gamma.|V|^2.(\beta+1)^2$

When this reasoning is applied for all the m properties in P, Equation 22 computes the time cost of the graph learning steps.

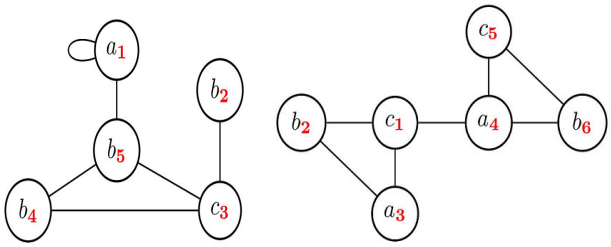
$$Time_1 = m.|V|^2.\left(p_{max} + \gamma.(\beta+1)^2\right) \tag{22}$$

The most time consuming operation realized during the vector computation step is the computation of the stationary distribution, which runs in around $r.(\beta+1)^3$ for the model associated with each property as stated in Section III-D. When all the m properties in P are considered, Equation 23 gives the time cost of the vector computation.

$$Time_2 = m.r.(\beta+1)^3 \tag{23}$$

In the worst case, the overall theoretical time cost required by the proposed approach for deriving the final feature vector $\mu_P(x)$ associated with a graph x according to a set P containing m properties is finally obtained in Equation 24 by summing the values computed in Equation 22 and in Equation 23.

$$\begin{aligned} Time &= Time_1 + Time_2 \\ &= m.|V|^2.\left(p_{max} + \gamma.(\beta+1)^2\right) + m.r.(\beta+1)^3 \end{aligned} \tag{24}$$



(a) The graph x (b) The graph $\overline{x}$

FIGURE 14. The graphs x and $\overline{x}$ compared in Table 6.

V. EXPERIMENTAL RESULTS

This section aims on the one hand at experimentally demonstrating the flexibility and the interpretability of the proposed approach compared to existing techniques, through a practical example of comparison of two graphs according to several sets of properties. On the other hand, the current section also aims at demonstrating that the proposed approach can exhibit better classification performances than existing techniques when the suitable set of properties is selected. All our source codes were written in C language and are available online[42] alongside with the corresponding experimental data. Undirected graphs are used in this section, but the proposed approach is also applicable to directed graphs.

A. EXPERIMENTAL SETTINGS

All the experiments performed in the current work were realized on a personal computer having 16 GB of main memory and the following processor: *Intel(R) Core(TM) i7-8665U CPU @ 1.90 GHz 2.11 GHz*. The constant ($r = 100$) was selected for computing the stationary distributions of each HMM and the maximum number of iterations ($\gamma = 100$) was selected during the HMMs training following [35].[43]

B. PRACTICAL EXAMPLE

Consider the vertex-labeled graphs x of Figure 14a (initially shown in Figure 2c) and $\overline{x}$ depicted in Figure 14b, such that the set of vertex labels is $\sum = \{a, b, c\}$. These two graphs have been chosen because they are small and simple, they thus enable an easy-to-understand explanation the proposed approach. However, the proposed approach also handles large and more complex graphs as it is demonstrated in Section V-C where large graph databases are experimented. When the proposed technique is applied on these two graphs considering the sets $\{p_3\}$, $\{p_7\}$, $\{p_8\}$ of properties described in Table 3 and using Equation 11 with the value ($M = 5$), the vectors contained in Table 5 are obtained for ($1 \leq \beta \leq 3$). The process of computation of each vector appearing in that table required in practice less than 10 milliseconds on the experimental computer. A simple vector concatenation enables to derive the vector corresponding to any mixture of these sets as described in Equation 20.

[42] https://webperso.etis-lab.fr/sylvain.iloga/index.html and https://github.com/MhdMrd/Flexible-graph-comparison-using-HMMs

[43] See Section V-B of [35].

TABLE 5. Feature vectors associated with the graphs x and $\overline{x}$ respectively shown in Figures 14a and 14b.

Vector	β	P		
		$\{p_3\}$	$\{p_7\}$	$\{p_8\}$
$\mu_P(x)$	1	$[0.31, 0.69]$	$[0, 0.83, 0.17]$	$[0.41, 0.28, 0.31, 0, 0, 0]$
	2	$[0.31, 0.69]$	$[0, 1, 0]$	$[0, 0.43, 0.57, 0, 0, 0]$
	3	$[0.69, 0.31]$	$[0.45, 0.55, 0]$	$[0, 0.33, 0.67, 0, 0, 0]$
$\mu_P(\overline{x})$	1	$[0.62, 0.38]$	$[0.36, 0.25, 0.39]$	$[1, 0, 0, 0, 0, 0]$
	2	$[0.80, 0.20]$	$[0.42, 0.22, 0.36]$	$[1, 0, 0, 0, 0, 0]$
	3	$[1, 0]$	$[0.38, 0.29, 0.33]$	$[1, 0, 0, 0, 0, 0]$

TABLE 6. *Euclidean* ($\tilde{d}$) and *Manhattan* ($\hat{d}$) distances between the graphs x and $\overline{x}$ respectively shown in Figures 14a and 14b.

Distance	β	P						
		$\{p_3\}$	$\{p_7\}$	$\{p_8\}$	$\{p_3, p_7\}$	$\{p_3, p_8\}$	$\{p_7, p_8\}$	$\{p_3, p_7, p_8\}$
$\tilde{d}_P(x, \overline{x})$	1	0.438	0.717	0.722	0.840	0.845	1.018	1.108
	2	0.692	0.956	1.228	1.180	1.410	1.556	1.704
	3	0.438	0.425	1.248	0.611	1.322	1.318	1.389
$\hat{d}_P(x, \overline{x})$	1	0.62	1.16	1.18	1.78	1.8	2.34	2.96
	2	0.98	1.56	2.0	2.54	2.98	3.56	4.54
	3	0.62	0.66	2.0	1.28	2.62	2.66	3.28

When the *Euclidean* and the *Manhattan* distances are used for comparing these two graphs according to the all the possible mixtures of the aforementioned sets of properties, the results contained in Table 6 are obtained. The content of Table 6 is the experimental proof of a fundamental observation made at the Introduction of the current paper which stated that: '*the result $d(x, \overline{x})$ of the comparison between two items x and $\overline{x}$ according to a metric d is not unique, but rather depends on the set P of comparison criteria*'.

Table 5 reveals that the proposed approach updates the feature vectors when β changes. This demonstrates that the information embedded in the actual neighborhood $\mathcal{G}(u)$ of each vertex u is accurately captured according to the variations of β. One can for example observe the gradual readjustment of the vectors $\mu_P(x)$ and $\mu_P(\overline{x})$ for $P = \{p_3\}$ when β changes, which materializes the fact that the parity of the degree of each vertex found in the actual neighborhood is accurately captured.

Table 5 also illustrates the capability of the proposed approach to effectively capture the overall adherence of each graph to the properties in P. One can for example observe that for $P = \{p_8\}$, the vector $\mu_P(\overline{x}) = [1, 0, 0, 0, 0, 0]$ is always obtained, irrespective of the value of β. This means that after a sufficiently long time, the model $\lambda_8(\overline{x})$ always spends 100% of its time observing the percentage 0%, irrespective of the depth of the vertex from which this observation is realized. This interpretation is accurate because in the graph $\overline{x}$, no vertex shares the same label with any of its immediate neighbors.

Another important information revealed by Table 5 is related to the fact that the proposed approach cannot be reduced to a simple statistical counting of the various values of each property appearing in the graph. It is effectively the behavior of the underlying HMM after a sufficiently long time that is considered. As an example, the vector $\mu_P(x) = [0, 1, 0]$ is generated for $P = \{p_7\}$, when $\beta = 2$. This results means that after a sufficiently long time when $\beta = 2$, the model $\lambda_7(x)$ always spends 100% of its time observing the label b. In other words, given that the label b '*regularly*' appears in x, the model $\lambda_7(x)$ decides to exclusively focus on this symbol after a sufficiently long time, and consequently stops observing the symbols a and c that '*rarely*' appear in x. However, the underlying HMM ignores only a when $\beta = 1$ and only c when $\beta = 3$.

TABLE 7. Statistics related to the original datasets *IMDB-BINARY* and *IMDB-MULTI* from the *TUDataset* repository.

Dataset	Number of classes	Class size	Max. per graph	
			Vertices	Edges
IMDB-BINARY	2	500	136	1249
IMDB-MULTI	3	500	89	1467

C. CLASSIFICATION EXPERIMENTS

All the classification experiments were realized with the software *WEKA* [41] (version 3.9) through a 10 fold cross-validation. *Support Vector Machines* with polynomial kernel (called *SMO* in *WEKA*) were used as classifier with their default parameters. The HMMs were generated for $(1 \leq \beta \leq 5)$. and feature vectors generated during each experiment were saved into online available '*.arff*' files[42] taken as inputs by *WEKA*. Equation 11 is used with the value ($M = 10$) during these classification experiments.

1) EXPERIMENTAL DATABASES AND SELECTED PROPERTIES

Classification experiments were initially conducted on the two following real-world datasets from the *TUDataset* repository[6]: *IMDB-BINARY* and *IMDB-MULTI*. These two datasets were created from the online database *IMDb*[44]

[44] www.imdb.com

TABLE 8. Properties verified by each vertex u of each graph x in the four modified datasets such that $l(u)$ and $f(u)$ are respectively the label and attribute of u. The value $\mathcal{C}(x) \in \{0, 1, 2\}$ is the class identifier of x in the dataset.

Dataset	Vertex label $l(u) \in \{1, 2, \ldots, 10\}$	Vertex attribute $f(u) \in \{-100, \ldots, 0, \ldots, 100\}$
IMDB-BINARY-vl	$(l(u)\%2) = (deg(u)\%2)$ **when** $\mathcal{C}(x) = 0$ $(l(u)\%2) \neq (deg(u)\%2)$ **when** $\mathcal{C}(x) = 1$	-
IMDB-BINARY-va	-	$(\lvert f(u)\rvert\%2) = (deg(u)\%2)$ **when** $\mathcal{C}(x) = 0$ $(\lvert f(u)\rvert\%2) \neq (deg(u)\%2)$ **when** $\mathcal{C}(x) = 1$
IMDB-MULTI-vl	$\mathcal{C}(x) = (l(u) + deg(u))\%3$	-
IMDB-MULTI-va	-	$\mathcal{C}(x) = (\lvert f(u)\rvert + deg(u))\%3$

TABLE 9. Experimental sets of properties selected for the four modified datasets. The values $l(u)$ and $f(u)$ are respectively the label and attribute of each vertex u of a graph x.

Dataset	P	Description
IMDB-BINARY-vl	$\{\tilde{p}_1\}$	$\tilde{p}_1 = [(l(u)\%2) \neq (deg(u)\%2)]$
IMDB-BINARY-va	$\{\tilde{p}_2\}$	$\tilde{p}_2 = [(\lvert f(u)\rvert\%2) \neq (deg(u)\%2)]$
IMDB-MULTI-vl	$\{\tilde{p}_3\}$	$\tilde{p}_3 = (l(u) + deg(u))\%3$
IMDB-MULTI-va	$\{\tilde{p}_4\}$	$\tilde{p}_4 = (\lvert f(u)\rvert + deg(u))\%3$

TABLE 10. Classification results for the original datasets when $P = \{p_1, \ldots, p_6\}$. Accuracies are in (%). The best accuracies are in bold.

Dataset	β				
	1	2	3	4	5
IMDB-BINARY	**74.0**	72.1	71.1	73.8	72.3
IMDB-MULTI	**51.13**	51.0	50.06	49.93	50.13

of information related to movies and television programs. They both contain unlabeled and unattributed graphs corresponding to movie collaborations and organized into classes. In these graphs, vertices represent actors/actresses and there is an edge between two vertices if the corresponding actors/actresses appear in the same movie. Statistics related to these two datasets are available in Table 7. The set $P = \{p_1, \ldots, p_6\}$ composed of 6 arbitrary properties described in Table 3 is selected for this first experiment, this induces feature vectors having 117 components considering the experimental settings.

Other classification experiments were subsequently conducted on four additional datasets derived from *IMDB-BINARY* and *IMDB-MULTI* by randomly adding either one attribute or one label to each vertex of the original dataset. In these modified datasets, each label belongs to the finite set $\sum = \{1, 2, \ldots, 10\}$ and each attribute is an integer belonging to the finite set $\{-100, \ldots, 0, \ldots, 100\}$. The properties verified by label $l(u)$ or the attribute $f(u)$ of each vertex u of each graph x in these four additional datasets are described in Table 8. Consequently, the sets $\{\tilde{p}_1\}$, $\{\tilde{p}_2\}$, $\{\tilde{p}_3\}$ and $\{\tilde{p}_4\}$ containing properties described in Table 9 are selected for this second experiment.

2) CLASSIFICATION RESULTS

Classification results for the two original datasets when $P = \{p_1, \ldots, p_6\}$ are presented in Table 10. According to this table, although only arbitrary properties have been selected, the feature vectors generated by the proposed approach enable to derive best classification accuracies of 74.0% for *IMDB-BINARY* and of 51.13% for *IMDB-MULTI*, both obtained with $\beta = 1$. However, the proposed approach always exhibited a perfect classification accuracy of 100% for each modified dataset according to its corresponding set of property, irrespective of the value of β. This perfect performance is due to the use of the most relevant and informative sets of properties during these experiments.

TABLE 11. Time cost of the classification experiments for each original dataset. Durations are in seconds.

Dataset	β				
	1	2	3	4	5
IMDB-BINARY	27	53	90	133	183
IMDB-MULTI	28	45	69	91	119

3) EXPERIMENTAL TIME COST

The experimental time cost required for deriving the feature vectors of each original dataset are available in Table 11. Less than 8 seconds were required for deriving the feature vectors of each modified dataset. These experimental time costs confirm that in practice, the proposed approach is seriously less time consuming than the existing GKs/GNNs, which generally required several minutes and sometimes required many hours for the same purpose.[45]

4) COMPARISON TO EXISTING TECHNIQUES

In this section, the performances of the proposed approach are only compared to those exhibited by GKs and GNNs, which are the most popular and the most efficient among the existing techniques. To achieve this goal, we first collected the best existing performances of GKs [1][46] and GNNs [15][47] for the two original datasets respectively exhibited by variants of the *Weisfeiler-Lehman* kernel and by the (GIN)-ϵ. Comparisons for the four modified datasets required new experiments involving GKs and GNNs. Therefore, we downloaded open access Python codes that implement classification using various GKs[48] and using the (GIN)-ϵ.[49] These Python codes were used without significant modifications except

[45] See columns 2 and 3 of the Table 8 in [1].
[46] See Table 7 of [1].
[47] See Table 1 of [15].
[48] https://ysig.github.io/GraKeL/0.1a8/auto_examples/
[49] https://github.com/chrsmrrs/tudataset/blob/master/tud_benchmark/

TABLE 12. Comparison results. Accuracies are in (%). The following acronyms are used for GKs: WL-PM = *'Weisfeiler-Lehman Pyramid Match'*, WL-VH = *'Weisfeiler-Lehman Vertex Histogram'*, NH = *'Neighborhood Hash'* and ML = *'Multiscale Laplacian'*. The best accuracies are in bold.

Dataset	This work		Existing work				Accuracy gain
	HMMs		GKs		GNNs		
	P	Acc.	Type	Acc.	Type	Acc.	
IMDB-BINARY	$\{p_1, \ldots, p_6\}$	74.0	WL-PM	73.6	(GIN)-ϵ	**75.1**	−1.1
IMDB-MULTI		51.13	WL-VH	51.7		**52.30**	−1.17
IMDB-BINARY-vl	$\{\tilde{p}_1\}$	**100**	NH	89.20		70.67	+10.8
IMDB-BINARY-va	$\{\tilde{p}_2\}$		ML	50.45		69.78	+30.22
IMDB-MULTI-vl	$\{\tilde{p}_3\}$		WL-VH	75.84		53.92	+24.16
IMDB-MULTI-va	$\{\tilde{p}_4\}$		ML	34.42		45.62	+54.38

the number of epochs of the (GIN)-ϵ that was reduced to 10 epochs due to its huge time cost for each modified dataset. Indeed, the experiments using the (GIN)-ϵ with 20 epochs were interrupted after more than one day of execution time without results.

Table 12 contains the best classification performances obtained after performing a 10-fold cross-validation using the (GIN)-ϵ and using various applicable GKs. It is important to mention that several hours were required for each of these classification experiments using some GKs and using the (GIN)-ϵ, despite of its small number of epochs. Table 12 reveals that the proposed approach is slightly beaten by the existing techniques for the two original datasets. This does not means that the proposed approach cannot outperform these existing techniques for these datasets. It rather means that the arbitrary properties used during these experiments are not enough suitable for distinguishing the classes found in these datasets. Table 12 also reveals that the proposed approach seriously outperforms GKs and GNNs for all the modified datasets with accuracy gains reaching +54.38% for the dataset *IMDB-MULTI-va*.

D. MAIN ASSETS OF THE PROPOSED APPROACH

The technique proposed in the current paper:

1) Requires the explicit specification of a finite set P of properties according to which the comparison is performed. No existing technique offers this flexibility. As a consequence, it can compare unlabeled and unattributed graphs, as well as labeled and attributed graphs depending on the nature of the properties in P.
2) Considers both, the local environment of each vertex, as well as the global structure of the graph. Indeed, for each property $p_i \in P$, the adherence of the vertices/edges found in the local neighborhood $\mathcal{G}(u)$ of each vertex u to the property p_i is first captured into a MC. The resulting MCs constructed for all the vertices of the graph are then learned by the *Baum-Welch* algorithm. As a consequence, unlike GKs that prioritize a predefined and limited set of graph properties, the proposed approach is suitable for comparing all kinds of graphs (small, large, dense, etc.).
3) Only requires a reasonable time cost (few minutes) during each experiment realized in the current paper, unlike GNNs and some GKs that require a huge time cost (several hours) for the same purpose. The proposed approach can also be easily implemented in parallel according to its methodology shown in Figure 10 in order to further reduce its time cost.
4) Unlike GNNs, it derives a HMM $\lambda_i(x)$ associated with a graph x (for each property $p_i \in P$) whose parameters can be easily modified and the resulting feature vector $\mu_i(x)$ is interpretable as it was demonstrated in Section V-B. Indeed, the l^{th} component $\mu_i^l(x)$ of $\mu_i(x)$ is the overall proportion of time spent by the model $\lambda_i(x)$ observing the l^{th} symbol after a sufficiently long time, irrespective of the state from which this observation is realized.
5) Seriously outperforms existing techniques for the flat classification of the modified experimental datasets with accuracy gains reaching +54.38% when the suitable sets of properties are selected.

VI. CONCLUSION AND PERSPECTIVES

The current paper tackles the problem of graph comparison which is an important tool for decision support, while being crucial for classification and clustering. Existing techniques used for this purpose (SSBTs, MBTs, GKs and GNNs) are limited by several factors stated in Section II-F and do not allow the explicit specification of user-defined properties on which the comparison should rely.

This paper attempts to attenuate the drawbacks of the existing techniques for graph comparison by proposing a flexible HMM-based technique that associates one feature vector with each graph while offering the explicit possibility to precise a finite user-defined set of properties that must be considered during the comparison. The comparison between two graphs is performed through the comparison of their associated feature vectors using any valid vector distance/similarity. Classification experiments conducted on two publicly available real-world graph datasets and on four synthetic datasets (derived from the two original real-world datasets) demonstrated that, when the suitable sets of properties are selected, the proposed approach outperforms existing techniques with accuracy gains reaching +54.38%.

The following perspectives can be considered in future work:

1) The proposed approach is highly customizable according to many parameters: $\beta, \gamma, r, M, \ldots$ Future work can analyze the impact of the gradual modification of these parameters on the performances of the proposed approach.

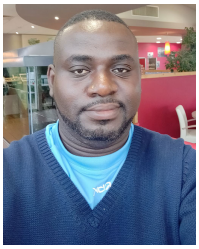
2) When it is required by the final goal of the comparison, future work can also focus on the automatic discovery of the most relevant and informative content of the set P of properties using for example machine learning or data mining techniques.
3) Future work can compare two finite graph sets by combining the proposed approach and the technique proposed in [35] for comparing two finite sets of vectors using HMMs. Indeed, given a finite set G of graphs, the proposed approach can be first used for generating one feature vector associated with each graph in G. The resulting set of vectors can then be associated with one unique feature vector using the technique proposed in [35]. Finally, the comparison between two graph sets is realized through the comparison of their associated feature vectors using any valid vector metric.
4) Another alternative for comparing two graph sets that are densely populated in future work consists in combining the proposed approach and the technique proposed in [42] for comparing two finite metric spaces using HMMs. Indeed, given a set P of properties, when a finite graph set G is attached to the distance d_P between two graphs proposed in the current paper, the resulting pair (G, d_P) is a finite metric space. Therefore, the technique proposed in [42] can be used for comparing two graph sets G_1 and G_2 through the comparison of the two finite metric spaces (G_1, d_P) and (G_2, d_P).
5) Future work may also analyze the possibility of designing a more robust version of the proposed approach that will first use different values of β for the same property, the resulting feature vectors will later be merged according to various principles into one unique vector.
6) The time cost of the proposed approach can be reduced in future work if the graph learning steps appearing in Figure 10 are performed in parallel for each graph. More precisely, if the set of properties is $P = \{p_1, \ldots, p_m\}$, then m processors $\{proc_1, proc_2, \ldots, proc_m\}$ can be used for this purpose for each graph, each processor $proc_k (1 \leq k \leq m)$ being responsible of the graph learning associated with p_k. The time cost can be further reduced if a parallel version of the *Baum-Welch* algorithm is executed. This can be implemented using *Message Passing Interface* (MPI) following [43] or using a *Field-Programmable Gate Array* (FPGA) chip following [44].

REFERENCES

[1] G. Nikolentzos, G. Siglidis, and M. Vazirgiannis, "Graph kernels: A survey," *J. Artif. Intell. Res.*, vol. 72, pp. 943–1027, Nov. 2021.
[2] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Apr. 2021.
[3] K. Filippova, "Multi-sentence compression: Finding shortest paths in word graphs," in *Proc. 23rd Int. Conf. Comput. linguistics (Coling)*, 2010, pp. 322–330.
[4] K. Jajuga, A. Sokolowski, and H.-H. Bock, *Classification, Clustering, and Data Analysis: Recent Advances and Applications* (Studies in Classification, Data Analysis, and Knowledge Organization). Berlin, Germany: Springer-Verlag, 2013.
[5] G. W. Milligan and S. C. Hirtle, "Clustering and classification methods," in *Handbook of Psychology: Research Methods in Psychology*, vol. 2. Hoboken, NJ, USA: Wiley, 2013, pp. 189–210.
[6] C. Donnat and S. Holmes, "Tracking network dynamics: A survey using graph distances," *Ann. Appl. Statist.*, vol. 12, no. 2, pp. 971–1012, Jun. 2018.
[7] D. G. Corneil and C. C. Gotlieb, "An efficient algorithm for graph isomorphism," *J. ACM*, vol. 17, no. 1, pp. 51–64, Jan. 1970.
[8] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 3, pp. 353–362, May 1983.
[9] M. A. Eshera and K.-S. Fu, "A graph distance measure for image analysis," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-14, no. 3, pp. 398–408, May 1984.
[10] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognit. Lett.*, vol. 19, nos. 3–4, pp. 255–259, Mar. 1998.
[11] M.-L. Fernández and G. Valiente, "A graph distance metric combining maximum common subgraph and minimum common supergraph," *Pattern Recognit. Lett.*, vol. 22, nos. 6–7, pp. 753–758, May 2001.
[12] P.-A. Champin and C. Solnon, "Measuring the similarity of labeled graphs," in *Proc. Int. Conf. Case-Based Reasoning, (ICCBR)*Trondheim, Norway. Cham, Switzerland: Springer, Jun. 2003, pp. 80–95.
[13] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, "A measure of similarity between graph vertices: Applications to synonym extraction and web searching," *SIAM Rev.*, vol. 46, no. 4, pp. 647–666, Jan. 2004.
[14] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Appl. Netw. Sci.*, vol. 5, no. 1, pp. 1–42, Dec. 2020.
[15] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–7. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km
[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
[17] S. Iloga, A. Bordat, J. Le Kernec, and O. Romain, "Human activity recognition based on acceleration data from smartphones using HMMs," *IEEE Access*, vol. 9, pp. 139336–139351, 2021.
[18] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "TUDataset: A collection of benchmark datasets for learning with graphs," 2020, *arXiv:2007.08663*.
[19] J. A. Bondy et al., *Graph Theory With Applications*, vol. 290. London, U.K.: Macmillan, 1976.
[20] N. Deo, *Graph Theory With Applications to Engineering and Computer Science*. New York, NY, USA: Dover, 2017.
[21] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Ser.*, vol. 2, no. 9, pp. 12–16, 1968.
[22] M. Ipsen and A. S. Mikhailov, "Evolutionary reconstruction of networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 66, no. 4, Oct. 2002, Art. no. 046109, doi: 10.1103/physreve.66.046109.
[23] N. N. W. B. Apolloni, "An introduction to spectral distances in networks," in *Proc. 20th Italian Workshop Neural Nets Neural Nets (WIRN)*, vol. 226. Amsterdam, The Netherlands: IOS Press, 2011, p. 227.
[24] J. Gu, J. Jost, S. Liu, and P. F. Stadler, "Spectral classes of regular, random, and empirical graphs," *Linear Algebra Appl.*, vol. 489, pp. 30–49, Jan. 2016.
[25] G. Jurman, R. Visintainer, M. Filosi, S. Riccadonna, and C. Furlanello, "The HIM glocal metric and kernel for network comparison and classification," in *Proc. IEEE Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2015, pp. 1–10.
[26] S. Chen, B. Ma, and K. Zhang, "On the similarity metric and the distance metric," *Theor. Comput. Sci.*, vol. 410, nos. 24–25, pp. 2365–2376, May 2009.
[27] T. Gärtner, "A survey of kernels for structured data," *ACM SIGKDD Explorations Newslett.*, vol. 5, no. 1, pp. 49–58, Jul. 2003.
[28] D. S. Johnson, "The NP-completeness column," *ACM Trans. Algorithms*, vol. 1, no. 1, pp. 160–176, Jul. 2005.

[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. 34th Int. Conf. Mach. Learning*, vol. 70, 2017, pp. 1263–1272.
[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–7.
[31] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5453–5462.
[32] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, 2018, pp. 4800–4810. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/e77dbaf6759253c7c6d0efc5690369c7-Paper.pdf
[33] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018, pp. 4438–4445.
[34] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.
[35] E. D. Madiga and S. Iloga, "Enhancing vector comparison using HMMs," *IEEE Access*, vol. 11, pp. 96939–96953, 2023.
[36] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proc. IEEE*, vol. 109, no. 3, pp. 247–278, Mar. 2021.
[37] R. Saleem, B. Yuan, F. Kurugollu, A. Anjum, and L. Liu, "Explaining deep neural networks: A survey on the global interpretation methods," *Neurocomputing*, vol. 513, pp. 165–180, Nov. 2022.
[38] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Jan. 1989.
[39] J. Kang. (2021). *Stat 243: Stochastic Process*. [Online]. Available: https://bookdown.org/jkang37/stochastic-process-lecture-notes/lecture09.html
[40] S. Iloga, "Customizable HMM-based measures to accurately compare tree sets," *Pattern Anal. Appl.*, vol. 24, no. 3, pp. 1149–1171, Aug. 2021.
[41] H. W. Ian and F. Eibe. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. [Online]. Available: http://weka.sourceforge.net/
[42] S. Iloga, "Histogram-based comparison of metric spaces using HMMs," *Evol. Intell.*, vol. 17, no. 2, pp. 1005–1022, Apr. 2024.
[43] J. B. B. Noussi, M. T. Tchendji, and S. Iloga, "Parallel HMM-based similarity between finite sets of histograms," in *Proc. 4th Conf. Recherches Informatique (CRI)*, 2019, pp. 1–8.
[44] A. López-López A. Espinosa-Manzo and M. O. Arias-Estrada, "Implementing hidden Markov models in a hardware architecture," in *Proc. Int. Meeting Comput. Sci. (ENC)*, Aguascalientes, Mexico, vol. 2, 2001, pp. 1007–1016.

MOHAMMAD MOURAD ABDOULAHI received the master's degree from the Faculty of Sciences, University of Maroua, Cameroon, under the supervision of Sylvain Iloga, in 2023. He has been a Software Engineer with the National Advanced School of Engineering, University of Maroua, since 2020. He has been a Teacher of computer science with the Higher Teachers' Training College, University of Maroua, since 2021, where he is currently carrying out a teaching internship. His research interest includes data modeling using hidden Markov models.



SYLVAIN ILOGA received the Ph.D. degree in computer science from the University of Yaoundé 1, Cameroon, in January 2018. Since January 2010, he has been a Teacher with the Department of Computer Science, Higher Teachers' Training College, Maroua, Cameroon. From September 2017 to August 2019, he completed a research and teaching internship at IUT Cergy-Pontoise, Neuville University, France. In January 2019, he was qualified as an Associate Professor at Section 27 (Computer Science), France. Since October 2022, he has been a Teacher with ESIEE-IT, CCI Paris île-de, France. Beside this, he is currently an Associate Member with the Research Laboratory ETIS, UMR 8051, ENSEA, CNRS, CY Cergy Paris Université, Cergy, France. In November 2023, he was subsequently promoted to the rank of an Associate Professor in Cameroon. His research interests include data modeling through machine learning using hidden Markov models, with the final goal to design their implementation into reconfigurable architectures.