A novel method for temporal graph classification based on transitive reduction

Carolina Stephanie Jerônimo de Almeida, Zenilton Kleber Gonçalves Do Patrocínio Jr, Simon Malinowski, Silvio Jamil F. Guimarães, Guillaume Gravier

A novel method for temporal graph classification based on transitive reduction

Carolina Jerônimo*†, Zenilton K. G. Patrocínio Jr.*, Simon Malinowski†,
Silvio Jamil F. Guimarães* and Guillaume Gravier†
*ImScience – PUC Minas – Belo Horizonte, Brazil
Email: carolinajeronimo@gmail.com, sjamil@pucminas.br, zenilton@pucminas.br
†IRISA – Université de Rennes, CNRS, Inria – Rennes, France
Email: simon.malinowski@irisa.fr, guig@irisa.fr

***Abstract*—Domains such as bio-informatics, social network analysis, and computer vision, describe relations between entities and cannot be interpreted as vectors or fixed grids, instead, they are naturally represented by graphs. Often this kind of data evolves over time in a dynamic world, respecting a temporal order being known as temporal graphs. The latter became a challenge since subgraph patterns are very difficult to find and the distance between those patterns may change irregularly over time. While state-of-the-art methods are primarily designed for static graphs and may not capture temporal information, recent works have proposed mapping temporal graphs to static graphs to allow for the use of conventional static kernels and graph neural approaches. In this study, we compare the transitive reduction impact on these mappings in terms of accuracy and computational efficiency across different classification tasks. Furthermore, we introduce a novel mapping method using a transitive reduction approach that outperforms existing techniques in terms of classification accuracy. Our experimental results demonstrate the effectiveness of the proposed mapping method in improving the accuracy of supervised classification for temporal graphs while maintaining reasonable computational efficiency.**

***Index Terms*—Temporal Graph, learning on dynamic graphs, temporal graph classification.**

I. Introduction

Graphs are used to represent linked data in various domains such as social network [1], [2], disease analysis [3], [4], [5], bioinformatics [6] and computer vision [7]. Modeling problems in terms of graphs allows for a deep understanding of the relationship between all elements. In social networks, for example, we can estimate how much a part of the network influences another [8]. In graph theory, a vertex is reachable from another if there exists, at least, a path between these two vertices. The existence of two or more different paths between two vertices, called the transitive relation, may be seen as redundant (or ambiguous) making it more difficult to understand the graph behavior. The result of the *Transitive Reduction* (TR) of a directed graph is a subgraph that has the same reachability relation as the original graph but with the minimum number of edges, preserving all paths between nodes in the original graph but removing unnecessary edges that can be inferred from the remaining edges [9]. An example of TR is shown in Figure 1, in which the edge *a–d* was removed because there is a path from *a* to *d* passing through *c* or *b*. TR is useful in many applications, such as database optimization, model checking, and software verification [10], [11], [12].

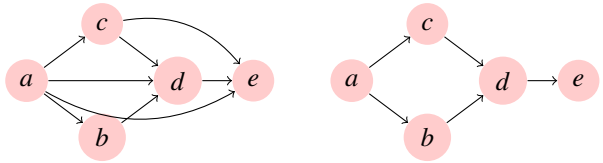

Fig. 1: Example of a transitive reduction in which the paths between all vertices are preserved. If there exists a path between two vertices in (a), then as can be seen in (b), there is one only path between them in the simplified graph.

Many real-world graphs are not *Directed acyclic graphs* (DAGs), for which TR is well-defined. Besides, most of those graphs have cycles, such as social networks and transportation networks, and TR may not be applicable or helpful in these cases. This motivates our decision to use TR for temporal graphs. When working with temporal graphs, since one cannot come back in time, it is possible to avoid cycles and be able to use the TR algorithm. And, as it will be shown, avoiding redundancy can improve the accuracy of classification tasks.

The idea of temporal graphs (also known as dynamic, evolving, or time-varying graphs), was first introduced in 1997 [13] discussing applications of temporal graphs and highlighting the great importance of a systematic treatment of the subject. The study in [14] provided tools and algorithms needed for controlling a system with dynamic graphs, the authors also studied on invariance and the reachability properties of these structures, proposing formalism leading to appropriate methods for dynamic graph problems. In an effort to integrate different notations found in the fields of delay-tolerant networks, the authors in [15] introduced the definition of Time-Varying Graphs. Later, a lot of works studied how algorithms for statics graphs can be applied with dynamics

The authors thanks Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – (PQ 306573/2022-9 and Universal 407242/2021-0), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES – (Grant COFECUB 88887.191730/2018-00) and Fundação de Amparo à Pesquisa do Estado de Minas Gerais – FAPEMIG – (Grants PPM-00006-18).

ones, such as shortest path [16], traveling salesman [17], and minimum spanning trees [18]. It is interesting to note that temporal graphs are present in numerous domains, such as communication networks [19], [20], [21], biological systems in cell and microbiology, neural networks [22], [23], and economics [24], [25]. In the literature, a lot of works have focused on how to apply learning tasks to such temporal graphs. Classification of the outbreak of fake news disseminated on social networks, the proliferation of disease, prediction of rerouting traffic, and video classification are examples of such learning tasks that can be applied to temporal graphs. Hence, while there are existing techniques for classifying temporal graphs, it is necessary to note that many of these methods are computationally expensive. This work addresses this issue by reducing this complexity while maintaining reasonable accuracy.

Therefore, the major goal of our work is to analyze the impact of the TR algorithm on temporal graphs for a classification task on real-world data sets, providing a performance comparison of different techniques in terms of accuracy and computational efficiency across different classification tasks. Furthermore, we introduce a novel method for mapping temporal graphs to static ones based on TR that outperforms existing techniques in terms of classification tasks. Experimental results demonstrate the effectiveness of the proposed approach in improving the accuracy of supervised classification for temporal graphs while maintaining reasonable computational efficiency. To the best of our knowledge, this work is the pioneer in the merging directed line graph and TR for classifying temporal graphs and our main contributions are the following: (i) a comprehensive analysis of how the removal of transitive edges affects the classification performance of temporal graphs; (ii) introduction of a novel mapping method of temporal graphs to static ones that outperforms existing techniques in terms of classification accuracy and running time; and (iii) a comprehensive evaluation using graph kernels on real-world data sets.

This work is organized as follows. In Section II, some related works are described. Section III presents the main concepts related to graph theory that are necessary for understanding the proposed method for classifying temporal graphs, which is described in Section IV. A quite extensive evaluation is given in Section V. And finally, some conclusions are drawn in Section VI.

II. Related work

In [26], the authors focus on the problem of evaluating the relation between events in a given text, which is important for information extraction. They argue that finding a common comparison referent at the text level is not straightforward and propose a shift from event-based measures to measures on a minimal underlying temporal graph, which is the TR of the graph of relations between event boundaries. They support their proposal by investigating its properties on synthetic data and a well-known temporal corpus. The authors aim to accomplish two things: to find a graph that is easy to compute and to eliminate the bias introduced by measures that do not consider the combinatorial aspect of agreement on transitive closure graphs. In [10], the authors proposed a method for analyzing directed acyclic graphs that take into account causality and highlight causal structure. The method is illustrated using citation networks from academic papers, patents, and US Supreme Court verdicts. The proposed approach is based on TR to remove unnecessary edges from a directed acyclic graph, which reveals the fundamental causal structure of the network. The authors have demonstrated how TR can identify differences in citation practices among different areas and can correct for the effect of a document's age on its citation count. Finally, the authors used TR to analyze null models of citation networks to illustrate the lack of causal structure in these models. Moreover, from the proposed method, it was possible to see that TR helped to reveal the causal skeleton of the network, after which standard network analysis tools may be used to analyze the network.

In [27], the authors have proposed a method for representing and analyzing temporal event data using weighted temporal event graphs, which are directed acyclic graphs where nodes represent events, and edges represent temporal orderings between the events. The weights of the edges capture the temporal distances between the events. To construct the graphs, the authors first constructed a directed graph of all possible temporal orderings between events. This graph can contain many transitive edges, which are edges that are implied by other edges in the graph and they used TR to simplify the graph and remove these transitive edges being easier to analyze and interpret. They also demonstrated the effectiveness of the weighted temporal event graphs approach on real-world datasets from social media and finance.

In [28], the authors have addressed the challenge of *de novo genome assembly* (computational biology), which involves decoding the sequence of an unknown genome from short sequences. They introduced new distributed-memory parallel algorithms for overlap detection and layout simplification steps of genome assembly. The algorithms are based on linear-algebra operations over semirings using 2D distributed sparse matrices, which reduces the need for different data structures in different steps of genome assembly. The proposed approach includes a novel distributed memory algorithm for the TR of the overlap graph, which simplifies the graph and makes it easier to resolve inconsistencies and create *contigs*.

Most related works have specific applications and datasets in mind, such as information extraction from text, citation networks, and social media/finance data. In contrast, this work aims to provide a more generalized approach applicable to a wider range of classification tasks on real-world datasets. This includes the exploration of mapping temporal graphs to static graphs and assessing the impact of transitive reduction, which surpasses existing techniques in terms of classification accuracy.

III. Fundamental concepts

Several definitions to model formally discrete temporal graphs are proposed in the literature. In [29], timed evolving graphs are defined as a system G' composed by a graph $G = (V, E)$, an ordered sequence of its subgraphs $S_g = (G_1, G_2, \cdots, G_t)$, then $G' = (G, S_g)$. The edge weights represent the traversal time. This notation is useful when one wants to predict the topology dynamics at different time intervals since the paths are restricted to never move into edges that existed only in past subgraphs. Similar to the previous definition, the authors in [30] defined a temporal graph by a sequence of time windows of snapshots of the network at that time interval. This work uses the definition proposed in [21] considering temporal graphs with edges existing at specific integral points in it and node labels. This definition allows us to model the problem, transforming it into static graphs to apply transitive reduction.

State-of-the-art techniques for temporal graph classification make use of graph kernels to compare different graphs. When graphs are represented as sets of features, such as node or edge attributes, important structural information about the graphs is lost, such as the presence of cycles, paths, or other higher-level patterns. Graph kernels, on the other hand, capture this structural information by computing a similarity score between pairs of graphs based on the graph structure, without requiring explicit feature representations. However, classical graph kernels can only be applied to static graphs. For temporal graph classification, temporal graphs are first mapped into static graphs before applying the kernels for classification. There are some mapping methods shown in [31], [32], and [16]. One of the best methods, in terms of temporal graph classification, to map temporal graphs into static graphs is called Direct Line Graph [33].

The fundamental concepts about temporal graphs, graph kernels, transitive reduction, and directed line graph are given in this section.

A. Temporal Graph

Let $G = (V, E)$ be an undirected (static) graph in which V is a finite set of vertices and E a finite set of undirected edges defined by $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. A labeled, undirected (static) graph (G, l) is a pair of an undirected graph and a labeling function $l : V \cup E \mapsto \Sigma$ that assigns a label to each vertex or edge of G, in which Σ is a finite alphabet. In a directed graph the set of edges E is defined by $E \subseteq \{(u, v) \in V \times V \mid u \neq v\}$. A (static) walk in a graph G is an alternating sequence of vertices and edges connecting consecutive vertices, but for simplification of the notation, we omit the edges. The length of a walk $(v_1, v_2, \cdots, v_{k+1})$ is k.

Now, let $\mathbf{G} = (V, \mathbf{E})$ be a temporal graph in which V is a finite set of vertices and $\mathbf{E}$ is a finite set of undirected temporal edges $e = (\{u, v\}, t)$ with u and v in V, $u \neq v$ and the availability time (or time stamp) $t \in \mathbb{N}$. A labeled, undirected, temporal graph $\mathbf{G} = (V, \mathbf{E}, l')$ consists of a temporal graph $\mathbf{G} = (V, \mathbf{E})$ and a labeling function $l' : V \cup T \mapsto \Sigma$ that assigns a label to each vertex at each time step $t \in T = 1, \ldots, t_{\max} + 1$ with $t_{\max}$ being the largest timestamp of any $e \in \mathbf{E}$.

For a temporal graph the number of edges is not polynomially bounded by the number of vertices. *A temporal walk* of length k is an alternating sequence of vertices and temporal edges $(v_1, e_1 = (v_1, v_2, t_1), v_2, \cdots, e_k = (v_k, v_{k+1}, t_k), v_{k+1})$ such that $t_i < t_{i+1}$ for $1 \leq i < k$. Moreover, for a temporal walk, the waiting time at vertex v_i with $1 < i \leq k$ is $t_i - (t_{i-1} + 1)$. The set of temporal walks (of length k) in a temporal graph $\mathbb{G}$ is denoted by $W_{tmp}(\mathbb{G})$ ($W^k_{tmp}(\mathbb{G})$). Finally, we define the function L that maps a temporal walk w to the label sequence $L(w) = (l(v_1, t_1), l(v_2, t_1 + 1), l(v_2, t_2), l(v_3, t_2 + 1), \cdots, l(v_k, t_k), l(v_{k+1}, t_k + 1))$.

B. Transitive Reduction

Let $G = (V, E)$ be a directed graph with vertex set V and edge set E. A (directed) path from vertex u to vertex v in a graph G is a walk from u to v without repetition of vertices. The TR of G is another directed graph $G' = (V, E')$ such that:

- G' has the same vertex set V as G.
- For every pair of distinct vertices $u, v \in V$, if there is a directed path from u to v in G, then there is also a directed path from u to v in G'.
- G' has the minimum number of edges among all graphs satisfying the first two conditions.

To define E', we first define the relation R on V as follows: for $u, v \in V$, uRv if and only if there is a directed path from u to v in G. Then, the transitive closure of R is the smallest transitive relation R' on V such that $R \subseteq R'$. We can define the edge set $E' = \{(u, v) \in E : u \neg R' v\}$. In other words, E' contains only the edges in E that are not necessary for maintaining the reachability relation in G. The resulting graph $G' = (V, E')$ is the transitive reduction of G [9]. The transitive reduction is well-defined only for DAGs.

The complexity of the transitive reduction algorithm depends on the size and structure of the input graph. Generally, the algorithm has a worst-case time complexity of $\mathcal{O}(n^3)$, in which n is the number of vertices in the graph. However, several optimizations and heuristics can be applied to improve the performance of the transitive reduction algorithm in practice. For example, the algorithm can be modified to use a breadth-first search instead of a depth-first search, which reduces the worst-case time complexity to $\mathcal{O}(n^2 \log n)$ for sparse graphs. In addition, the algorithm can be parallelized to take advantage of multi-core processors, or specialized hardware such as GPUs, to further improve performance. A comparison (considering time and space complexity) of several algorithms on TR computation is presented in [12].

C. Graph Kernels

A graph kernel is a function $k : G \times G \mapsto \mathbb{R}$ that maps pairs of graphs to a real number representing the dissimilarity between two graphs. Graph kernels need to have some properties: (i) symmetry – $k(G_1, G_2) = k(G_2, G_1)$ for all graphs G_1 and G_2; (ii) positive semi-definiteness – $k(G_1, G_2) \geq 0$ for all

pairs of graphs G_1 and G_2, and $k(G_1, G_2) = 0$ if and only if $G_1 = G_2$; and (iii) compositional – for any function $f : V \mapsto \mathbb{R}$, the kernel $k_f(G_1, G_2) = \sum_{u,v \in V} f(u)f(v)k_G(u, v)$ is also a valid kernel, where $k_G(u, v)$ is the kernel value between nodes u and v in the graphs G_1 and G_2. We briefly summarize two well-known kernels for static graphs.

The *Graphlet kernel* computes the similarity between two graphs counting the occurrences of graphlets of different sizes. A k-graphlet is a connected subgraph H of G with k vertices, denoted by $H \in G_k$. More formally, let G be a graph with vertex set $V(G)$ and edge set $E(G)$, and let G_k be the set of all k-graphlets of G. The graphlet degree vector of G is a vector of counts of each k-graphlet in G, denoted by $g_k(G) = (g_1(G), g_2(G), ..., g_{|G_k|}(G))$. The graphlet kernel between two graphs G and H is then defined as the inner product of their graphlet degree vectors:

$$K(G, H) = \sum_k \Phi(k) g_k(G) \cdot g_k(H) \tag{1}$$

in which $\Phi(k)$ is a weighting function that assigns different weights to graphlets of different sizes, and $\cdot$ denotes the inner product between two vectors, see [34] for more.

The *Weisfeiler-Lehman subtree kernel* is a kernel function that computes the similarity between two graphs based on their shared subtrees. The Weisfeiler-Lehman subtree kernel proceeds iteratively, refining a labeling function for the vertices of each graph using information about the local neighborhood of each vertex [35]. At each iteration t, the labeling function $h_t : V \mapsto \mathbb{N}^k$ maps each vertex $v \in V$ to a vector of k integer labels that encode the frequency of certain local subtrees in the neighborhood of v.

$$k_{WL}(G_1, G_2) = \sum_{t=0}^{\infty} \alpha^t \sum_{v \in V} h_t^{(1)}(v) \cdot h_t^{(2)}(v) \tag{2}$$

in which $h_t^{(1)}(v)$ and $h_t^{(2)}(v)$ are the label vectors for vertex v in graphs G_1 and G_2 at iteration t, respectively. The parameter α is a damping factor that controls the weight given to iterations at different depths in the subtree hierarchy.

D. Directed Line Graph

A line graph of a static graph G is a graph whose vertices are the edges of G that are connected if they share a vertex in G. The authors in [21] used *DL* to encode the temporal information in a study of the dissemination process. Each temporal edge is represented by two vertices and a temporal walk can be performed, since a walk in the *DL* graph is related to the temporal walks in the original temporal graph having the same label sequence, being able to model waiting times and keeping the temporal information. Following [21], *Directed line graph expansion* (DL) can be defined as follows: given a temporal graph $(\mathbf{G}, l)$, the directed line graph expansion $DL(\mathbf{G}, l) = (\mathbf{G}', l')$ in which $\mathbf{G}' = (V', E')$ is the directed graph, where every temporal edge $(\{u, v\}, t)$ is represented by two vertices $n^t_{\overrightarrow{uv}}$ and $n^t_{\overrightarrow{vu}}$ and there is an edge from $n^t_{\overrightarrow{uv}}$ to $n^s_{\overrightarrow{xy}}$ if $v = x$ and $t < s$. For each vertex $n^t_{\overrightarrow{uv}}$, the label $l'\left(n^t_{\overrightarrow{uv}}\right) = (l(u, t), l(v, t + 1))$ is set. Figure 2 shows an example of the directed line transformation. The edge value is related to waiting time: if the previous node edges are consecutive (e.g., edges 2 and 3 in Figure 2a), then there is no waiting time between edges, represented by value 1, otherwise, 0 is used.

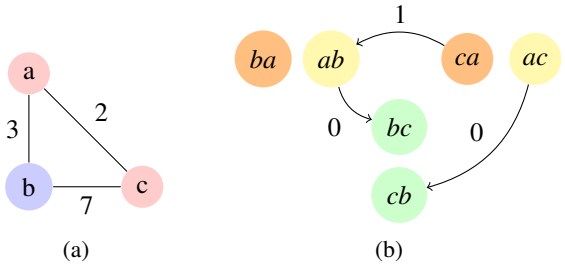

Fig. 2: Example of directed line transformation. The walk $(n^2_{ca}, n^3_{ab}, n^7_{bc})$ of length 2 in (b) corresponds to the temporal walk *(c,(c, a, 2), a,(a, b, 3), b,(b, c, 7), c)* of length 2 in the temporal graph (a) [21].

IV. Transitive Reduction on Temporal Graphs

For decreasing the size of graphs and keeping just one path between vertices, if there exists, we can apply transitive reduction. Here, we will study the impact of TR on temporal graphs. For that, temporal graphs are mapped to static graphs so that conventional static kernels, for example, *graphlet* or *Weisfeiler-Lehman subtree kernel* can be applied.

In [33], the authors have proposed different methods for mapping temporal graphs into static ones. The DL approach outperformed other methods capable of fully encoding the temporal information. As in the DL transformation the edges are vertices, and the number of vertices is $|E|$. The maximal number of edges is reached when each vertex of the original temporal graph for every incoming edge can be combined with all outgoing edges [33], then the number of edges in the DL is $O(|E|^2)$ leading to a quadratic blowup with regard to the number of temporal edges. Proposition 1 shows that the conversion of a temporal graph to DL leads to a DAG, then we are able to apply the TR algorithm to deal with the size of edges.

Proposition 1 (Directed line graph): Let $(\mathbf{G}, l)$ be a temporal graph. The resulting $DL(\mathbf{G}, l)$ is a *Directed acyclic graph* (DAG).

Proof: Suppose there exists a directed cycle in $DL(\mathbf{G}, l)$. Let $n^t_{\overrightarrow{uv}}$ be a vertex on the cycle with the earliest timestamp t. Let $n^s_{\overrightarrow{xy}}$ be the last vertex of the cycle that connects $n^t_{\overrightarrow{uv}}$. Since there is a path from $n^s_{\overrightarrow{xy}}$ to $n^t_{\overrightarrow{uv}}$, we know that $y = u$ and $s > t$. However, $n^t_{\overrightarrow{uv}}$ represents a temporal edge (u, v, t), and $n^s_{\overrightarrow{xy}}$ represents a temporal edge (x, y, s). Therefore, $u = y = x$ and $t > s$. But this contradicts the fact that $n^t_{\overrightarrow{uv}}$ has the earliest timestamp on the cycle. Therefore, there are no directed cycles in $DL(\mathbf{G}, l)$, and $DL(\mathbf{G}, l)$ is a DAG. ■

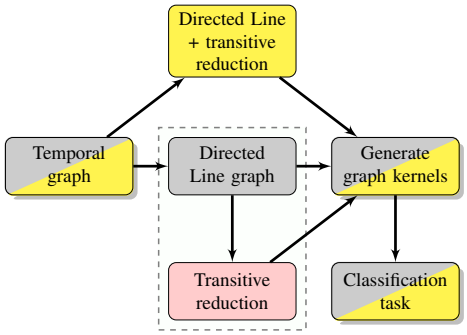

Fig. 3: Overview of the different temporal graph classification pipelines followed in this work.

To the best of our knowledge, this work proposes a novel methodology for temporal graph classification that makes use of directed line graph and TR. Figure 3 schematically represents the different pipelines used in this work to perform the classification task, yiedling three possibilities. The first one, called baseline [33], is represented by the gray blocks in Figure 3, in which temporal graphs are mapped to static graphs using the DL transformation. Then, the normalized Gram Matrix is computed using two graph kernels (Graphlet and Weisfeiler-Lehman subtree kernel) for the learning task. The second one, called TR-based method, is represented by gray and red blocks in Figure 3, we propose to apply TR after the DL transformation in order to reduce the size of these graphs. The impact of TR on classification accuracy and running time compared to *baseline* is analyzed in the experimental results. Finally, we propose also a method, called DLTR-based method (illustrated by yellow blocks), which is an extension of the TR-based method that combines the DL and TR methods. We detail below how this extension is realized.

There are different ways to remove transitive edges, but the TR algorithm used in the TR-based method removes all transitive edges, which is more complex, performing a depth-first search or breadth-first search on the graph, and computationally expensive for large graphs. Additionally, the algorithm requires the use of data structures such as stacks or queues to keep track of the visited nodes and the order in which they were visited. The concept in the DLTR-based method is to remove a certain amount of transitive edges using the simplest implementation of the TR algorithm. The basic idea of the TR algorithm is that for each set of three nodes $(x, y, z) \in V'$ if there is a path xy, yz we can remove xz if it exists. However, as shown in Proposition 2, it is impossible to have transitive edges with only three nodes, it is only possible with at least four nodes, as in Figure 4b. Having more than three loops can be time-consuming and does not guarantee the removal of transitive edges.

Proposition 2: Let $DL(\mathbf{G}, l)$ a directed line expansion of a temporal graph $(\mathbf{G}, l)$. If there exists a transitive edge in $DL(\mathbf{G})$, this transitive edge involves at least four consecutive nodes in $DL(\mathbf{G}, l)$.

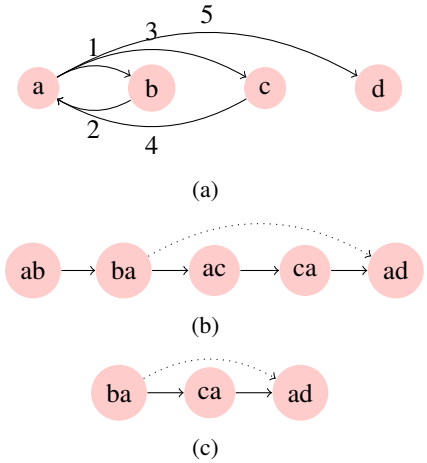

Fig. 4: Example of temporal graph (a), DL conversion in (b) and DL conversion with temporal graph smoothing (c). The dotted lines represent transitive edges that can be removed.

Proof: Let $v_1 = n^t_{\overrightarrow{uv}}$, $v_2 = n^s_{\overrightarrow{vx}}$ and $v_3 = n^p_{\overrightarrow{xv}}$ with $t < s < p$ be 3 connected vertices, $v_1, v_2, v_3 \in V'$ and $u, v, x \in V$. To create a transitive edge we need to connect v_1 with v_3. However, this leads to a contradiction, as v_1 corresponds to a temporal edge (u, v, t) and v_3 corresponds to a temporal edge (x, v, p) and they are not consecutive in $(\mathbf{G}, l)$ since the temporal edge v_3 starts with vertex x not v. To be able to construct a transitive edge in $DL(\mathbf{G}, l)$ we need to create a temporal edge starting with v and ending with u with a time higher than p. With this, we create a vertex $v_4 = n^n_{\overrightarrow{vu}}$, $n > p$ connected to v_3 and now we are able to create a transitive edge from v_1 to v_4. ■

To deal with this, we have used graph simplification (that we denote temporal graph smoothing), adapted from graph smoothing. Graph smoothing, also known as smoothing away or smoothing out, is the process of replacing edges incident at a vertex of degree 2 by a single new edge and removing the vertex. When running the DL algorithm, each temporal edge is transformed into a node. Before creating the node, for each edge $n^t_{\overrightarrow{uv}}$ we check if there exists a $n^p_{\overrightarrow{vu}}$ with $p > t$. This means that, instead of creating two consecutive nodes, we create just one that corresponds to the last temporal edge $n^p_{\overrightarrow{vu}}$. The label of a node in the directed line graph is a combination of the labels of its corresponding edges in the temporal graph as shown in section III-D. Maintaining the same idea, the node label that corresponds to the first temporal edge is combined with the one of the last node. For example, in Figure 4c the nodes *ab* and *ac* were not created since for *ab*, *ba* exists and for *ac*, *ca* exists.

The neighbors' edges from the first temporal edge are propagated to the last one to keep the structure. The result is a compact graph (see Figure 4c), and the TR algorithm can be applied to remove the transitive edges (dotted line in Figure 4c) in a much faster way. In the next section, we will study the impact of these two novel methods (TR- and DLTR-based methods) on both classification accuracy and efficiency.

V. Experiments

In this Section, we describe the experiments and the obtained results taking into account real-world datasets.

A. Datasets

In order to provide a comparative analysis between the different strategies described before, we have used six different databases provided by TUDataset [36]. This benchmark provides temporal graph classification datasets derived from *Tumblr*, *Dblp*, *Facebook* as well as contacts between students at MIT, in a *Highschool*, and visitors at the *Infectious* exhibition, for dissemination process study. For each dataset, a dissemination process simulation was done, in which nodes are infected at different time instants, providing two classification tasks for each dataset. The first task involves discriminating between temporal graphs with vertex labels resulting from a dissemination process and those without. To accomplish this, the authors ran a *Susceptible-infected* (SI) simulation with fixed parameters on half of the dataset and used it as the first class. The second class was made up of the remaining graphs. For each graph in the second class, the authors counted the number of infected vertices, reset the labels, and then randomly infected a number of vertices at a random time. The second task involves discriminating between temporal graphs that differ in the dissemination process itself. For this task, the authors ran the SI simulation with different parameters for each of the two subsets. For both subsets, $I = 0.5$ (initial infection rate), but for the first subset, the authors set $p = 0.2$ (infection probability), and for the second subset, $p = 0.8$. The simulation runs repeatedly until at least $|V| \times I$ vertices are infected or no more infections are possible, for example, if a graph has 100 vertices and the initial infection rate is set to 0.5, then initially 50 vertices are infected. In order to stop the SI simulation, at least $50 \times 0.5 = 25$ vertices (i.e., 50% of the total number of vertices) need to be infected. The simulation continues until either this condition is met or no more infections are possible.

B. Graph Kernels

As a baseline we use the 3-node Graphlet (GL) and the Weisfeiler-Lehman subtree (WL) kernels on static graphs obtained by interpreting the timestamps as discrete edge labels, and assigning to each vertex the concatenated sequence of its labels. The source code is provided by [36].

C. Experimental Setup

The normalized Gram matrix was calculated for each kernel and then we used the C-SVM implementation of LIBSVM [37] to determine the classification accuracies. We performed 10-fold cross-validation to select the C parameter from the range of $10^{-3}, 10^{-2}, ..., 10^{2}, 10^{3}$ on the training folds. We repeated the 10-fold cross-validation ten times with different random folds to obtain the average accuracies and standard deviations. The number of iterations for the Weisfeiler-Lehman subtree kernel (from 0 to 5) was selected through 10-fold cross-validation. A NetworkX implementation of a transitive reduction will be used in the TR-based method [38].

D. Evaluation and Discussion

Table I shows that TR algorithm improved the classification accuracy. This improvement is more significant when the dataset has a bigger quantity of edges (Highschool and MIT). Reducing the graph size can help to eliminate noise and irrelevant information from the data, making it easier for the SVM to identify the patterns and features that are truly relevant to the classification task. This can lead to a more accurate classification because the SVM is working with a more focused and relevant set of features. Additionally, reducing the graph size can also reduce the risk of overfitting, which occurs when a model becomes too complex and starts to fit the noise in the data instead of the underlying patterns [39]. Overfitting occurs when a model fits too closely to the training data and fails to generalize to new data. In the case of graph kernels, the size and complexity of the graph can affect the performance of the kernel function, and a smaller graph can improve the kernel's ability to generalize to new data by reducing the number of possible patterns or subgraphs that

TABLE I: Classification accuracy in percent and standard deviation for the first classification task. **DL**, is only the directed line approach, **DL + TR**, is the transitive reduction algorithm applied after the directed line transformation and **DLTR**, is the TR algorithm applied at the same time of DL.

	Datasets					
Kernel	Highschool	Infectious	Tumblr	Dblp	Facebook	MIT
Baseline						
DL-GL	93.83±0.8	97.05±0.8	89.37±1.0	96.39±0.4	92.31±0.3	OOM
DL-WL	97.44±0.4	98.60±0.3	93.59±1.0	98.42±0.3	95.89±0.2	OOM
DL + TR-based						
DL-GL	**96.94**±0.9	95.80±1.1	**92.33**±0.8	**96.91**±0.2	**93.74**±0.2	OOM
DL-WL	97.45±0.9	98.10±0.5	**93.88**±0.6	**98.90**±0.1	**96.07**±0.2	OOM
DLTR-based						
DL-GL	**95.77**±0.6	96.30±0.6	**93.26**±0.5	**97.23**±0.3	**94.32**±0.2	**84.04**±2.7
DL-WL	**98.11**±0.7	97.80±0.4	**93.50**±0.8	**98.60**±0.2	95.06±0.4	**87.85**±1.8

TABLE II: Classification accuracy in percent and standard deviation for the second classification task.

Kernel	Datasets					
	Highschool	Infectious	Tumblr	Dblp	Facebook	MIT
Baseline						
DL-GL	91.05±1.9	88.10±1.9	78.56±1.5	79.68±0.7	75.86±0.4	OOM
DL-WL	87.16±1.0	80.25±1.6	75.20±2.1	77.18±0.9	80.35±0.7	OOM
DL + TR-based						
DL-GL	**94.61**±1.0	83.85±1.8	**80.15**±1.1	**80.09**±0.7	74.35±0.3	OOM
DL-WL	**90.61**±1.5	**81.00**±1.2	**77.85**±0.8	**79.77**±0.8	**82.16**±0.7	OOM
DLTR-based						
DL-GL	88.22±0.8	82.45±1.6	**79.91**±1.4	78.33±0.6	72.81±0.5	**60.61**±2.7
DL-WL	**89.11**±1.1	79.90±2.4	**79.04**±0.9	**77.27**±0.4	77.20±0.6	**59.91**±4.2

can be learned from the training data. MIT is a large dataset with more than 62 million edges. When we tried to run the *Baseline* and *TR-based method* we got OOM (Out of memory), being able to run just the *DLTR-based method*.

For the second classification task, Table II shows that reducing the size of the graph also improves accuracy, but when we do the temporal smoothing process, the graphs lose some relevant information about the graph structure for classification. However, the accuracy of the DLTR-based method is very similar to the baseline when the overall quantity of edges is not high, leading to consider a slight loss of accuracy. In general, the second classification task poses a greater challenge for the temporal approaches that reach lower accuracy than the first classification task. Especially, the MIT data set seems to be hard. In Figure 5 we can see that TR improved accuracy compared to the baseline and when compared to the DLTR-based (see the third plot), the accuracy is quite similar, gaining in running time.

As we can see in Table III, the size of the graphs decreases when the transitive reduction algorithm is applied, which was expected. The reduction in size is most significant in the DLTR-based method, in which both the line graph conversion and transitive reduction are applied simultaneously. In the TR-based method, the reduction in edge count varies depending on the dataset, but it is significant in the case of the Tumblr and Highschool dataset, where the number of edges decreased from 412,892 to 205,357 and from 2,079,062 to 360,895. This is a reduction of almost 50% and 83% in edge count, respectively. The reduction in the number of vertices is zero, as transitive reduction does not affect the number of vertices in a graph. Compared to the TR-based method, the DLTR-based method resulted in a further reduction in the number of edges and vertices for all datasets. The reduction in the number of edges is most significant in the case of the Highschool dataset, where the number of edges decreased from 360,895 to 252,081, which is a further reduction of about 30%. The reduction in the number of vertices is also significant for all datasets, with reductions ranging from about 3% to almost 70% for the MIT dataset. Overall, with the DLTR-based method, we can significantly reduce their size while still maintaining their essential properties for classification tasks. The reduction in size can be more or less significant depending on the dataset, but it is generally more significant for larger graphs.

Table IV shows that the time spent to perform transitive reduction separately is significantly longer than the time necessary to convert the temporal graph to a directed line

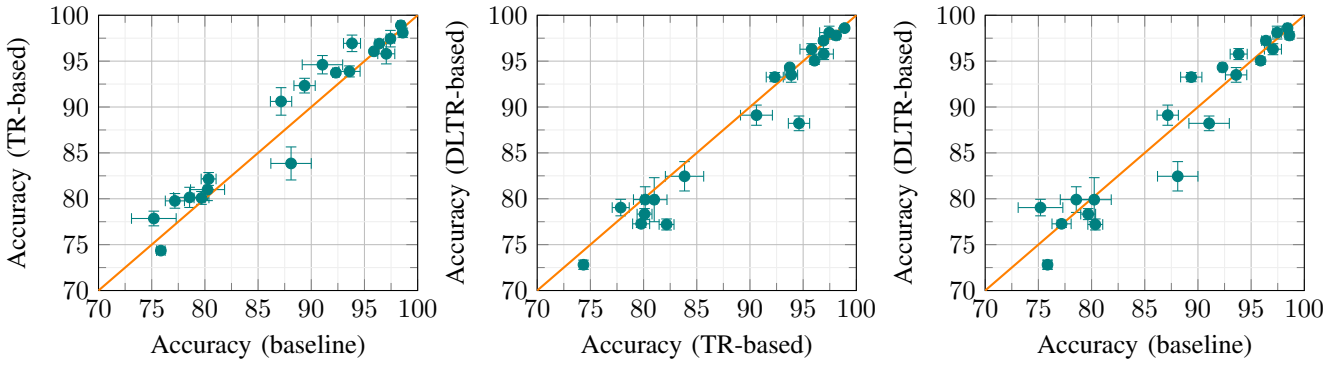

Fig. 5: Scatter plots to show the accuracy relationship between two different methods. The dots are accuracy in different datasets and using different graph kernels for classification.

TABLE III: Graph size in terms of the overall quantity of Vertices and Edges.

	Datasets					
Size	Highschool	Infectious	Tumblr	Dblp	Facebook	MIT
Baseline						
Sum $\|E\|$	2,079,062	918,513	412,892	1,097,872	851,165	62,923,589
Sum $\|V\|$	98,066	91,944	74,520	241,674	267,673	142,508
DL + TR-based						
Sum $\|E\|$	360,895	380,920	205,357	777,212	516,631	-
Sum $\|V\|$	98,066	91,944	74,520	241,674	267,673	142,508
DLTR-based						
Sum $\|E\|$	**252,081**	436,015	**165,119**	839,755	**531,431**	170,080
Sum $\|V\|$	**69,197**	**71,157**	**63,971**	**196,154**	**234,983**	**74,801**

graph in both baseline and DLTR-based. This is particularly evident in the Facebook dataset, in which transitive reduction takes over 9000 milliseconds while baseline and DLTR-based take only 90 and 248 milliseconds, respectively. However, it is important to note that the TR-based method takes longer than the baseline and DLTR-based method in all datasets. This is likely due to the additional step of performing transitive reduction on the already converted directed line graph. Overall, we can see that the DLTR-based method is the most efficient in terms of both accuracy and time, as it combines the conversion and transitive reduction steps, resulting in faster processing time and improved accuracy.

In Table V, the time required for the classification step of the three methods is presented. One can see that the DL-GL kernel is faster than the DL-WL kernel for all datasets and all methods. This happens because the Weisfeiler-Lehman kernel involves the computation of the neighborhood of nodes and updating labels iteratively, while the graphlet kernel only counts the number of isomorphic graphlet subgraphs, which could also contribute to the speed difference. Regarding the running times, the DLTR-based method is generally slower than the other two methods, which is also expected as it involves an additional step of transitive reduction and temporal graph smoothing. However, the difference in running times between the TR-based method and the DLTR-based method is not significant in most cases, except for the MIT dataset in which the DLTR-based method is much faster than TR-based. Overall, we can conclude that the DLTR-based method offers improved accuracy at the cost of longer running times compared to other methods, but the difference in running times is not significant in most cases.

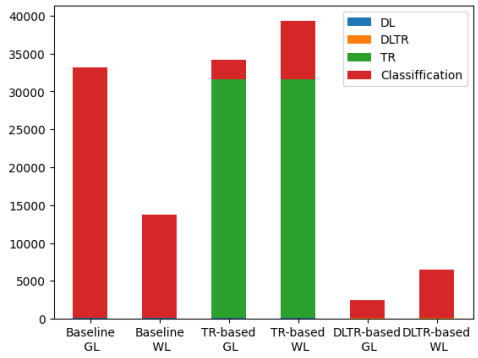

Fig. 6: Temporal Graph Classification time of Highschool data set (ms).

Based on Figure 6, it is evident that the overall time for temporal graph classification varies among different methods utilizing two different kernels. The mapping time for *DL* and DLTR appears negligible in comparison to the time consumed by classification and transitive reduction processes. Due to the minimal contribution of mapping time, the colors representing *DL* and DLTR are nearly imperceptible. Notably, the DLTR-based method outperforms the others in terms of speed, as indicated by the significantly lower classification time.

VI. Conclusion

This work presented a novel method for improving the classification accuracy of temporal graphs by combining line graph transformation and transitive reduction techniques. Experimental results demonstrate that reducing the size of the graph by removing redundant edges and nodes can lead to

TABLE IV: Mapping time in milliseconds for the baseline, the TR-based method plus TR and DLTR-based method running time.

	Datasets					
	Highschool	Infectious	Tumblr	Dblp	Facebook	MIT
DL + (TR)	126 + (31510)	68 + (15398)	28 + (2380)	115 + (4737)	90 + (9134)	4939 + (OOM)
DLTR	147	183	62	298	248	522

TABLE V: Time for classification performance of the three methods, in seconds.

	Datasets					
Kernel	Highschool	Infectious	Tumblr	Dblp	Facebook	MIT
DL						
DL-GL	33.07	9.08	4.67	12.44	16.36	–
DL-WL	13.57	9.22	11.41	32.84	56.06	–
DL + TR						
DL-GL	2.59	3.07	2.74	9.89	13.57	–
DL-WL	7.68	7.39	10.48	32.53	54.25	–
DLTR						
DL-GL	**2.29**	4.16	2.80	10.56	12.77	1.60
DL-WL	**6.39**	**7.01**	**10.04**	**30.98**	55.38	5.38

improved classification accuracy, especially for larger graphs, since real-world graph datasets can be incredibly large and complex, containing millions or even billions of nodes and edges like transportation networks, financial transaction networks, and communication networks. The proposed method achieved similar or better accuracy compared to the state-of-the-art methods while also being computationally more efficient. Moreover, the proposed method provides a significant reduction in the size of the graph, making it more manageable for further analysis and visualization. Finally, this work contributes to the development of graph-based machine learning methods for temporal graph data and can be applied to a wide range of temporal graph data sets, such as social networks and communication networks, to improve their classification performance. Thus, this work's findings have the potential to benefit a wide range of applications, from predicting disease outbreaks to detecting online fraud.

Future research opportunities in this area include practical implementation and application of different transitive reduction techniques to determine their real-world performance. Additionally, exploring the effectiveness of the method on graphs with multiple edge types or different structures and investigating its potential for classification tasks in temporal graph databases, such as video classification, where each video can be modeled as a temporal graph, can be valuable. In conclusion, the method proposed is a promising avenue for future research on temporal graph classification and has the potential to be further optimized and improved in future studies.

REFERENCES

[1] A. Amara, M. A. Hadj Taieb, and M. Ben Aouicha, "Multilingual topic modeling for tracking covid-19 trends based on facebook data analysis," *Applied Intelligence*, vol. 51, no. 5, pp. 3052–3073, 2021.

[2] N. Akhtar and M. V. Ahamad, "Graph tools for social network analysis," in *Research Anthology on Digital Transformation, Organizational Change, and the Impact of Remote Work*. IGI Global, 2021, pp. 485–500.

[3] A. Karaivanov, "A social network model of covid-19," *Plos one*, vol. 15, no. 10, p. e0240878, 2020.

[4] Y. Zhu, J. Ma, C. Yuan, and X. Zhu, "Interpretable learning based dynamic graph convolutional networks for alzheimer's disease analysis," *Information Fusion*, vol. 77, pp. 53–61, 2022.

[5] J. Wang, A. Ma, Y. Chang, J. Gong, Y. Jiang, R. Qi, C. Wang, H. Fu, Q. Ma, and D. Xu, "scgnn is a novel graph neural network framework for single-cell rna-seq analyses," *Nature communications*, vol. 12, no. 1, pp. 1–11, 2021.

[6] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang, "Graph neural networks and their current applications in bioinformatics," *Frontiers in genetics*, vol. 12, 2021.

[7] P. Pradhyumna, G. Shreya *et al.*, "Graph neural network (gnn) in image and video understanding using deep learning for computer vision applications," in *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*. IEEE, 2021, pp. 1183–1189.

[8] S. Yang, "Networks: An introduction by mej newman: Oxford, uk: Oxford university press. 720 pp., $85.00." 2013.

[9] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.

[10] J. R. Clough, J. Gollings, T. V. Loach, and T. S. Evans, "Transitive reduction of citation networks," *Journal of Complex Networks*, vol. 3, no. 2, pp. 189–203, 2015.

[11] V. Dubois and C. Bothorel, "Transitive reduction for social network analysis and visualization," in *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*. IEEE, 2005, pp. 128–131.

[12] X. Tang, J. Zhou, Y. Qiu, X. Liu, Y. Shi, and J. Zhao, "One edge at a time: A novel approach towards efficient transitive reduction computation on dags," *IEEE Access*, vol. 8, pp. 38 010–38 022, 2020.

[13] F. Harary and G. Gupta, "Dynamic graph models," *Mathematical and Computer Modelling*, vol. 25, no. 7, pp. 79–87, 1997.

[14] M. Mesbahi, "On a dynamic extension of the theory of graphs," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, vol. 2. IEEE, 2002, pp. 1234–1239.

[15] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 27, no. 5, pp. 387–408, 2012.

[16] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.

[17] O. Michail and P. G. Spirakis, "Traveling salesman problems in temporal graphs," *Theoretical Computer Science*, vol. 634, pp. 1–23, 2016.

[18] S. Huang, A. W.-C. Fu, and R. Liu, "Minimum spanning trees in temporal graphs," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 419–430.

[19] B. Tadić, "Dynamics of directed graphs: the world-wide web," *Physica A: Statistical Mechanics and its Applications*, vol. 293, no. 1-2, pp. 273–284, 2001.

[20] S. Ozcan, M. Astekin, N. K. Shashidhar, and B. Zhou, "Centrality and scalability analysis on distributed graph of large-scale e-mail dataset for digital forensics," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 2318–2327.

[21] L. Oettershagen, N. M. Kriege, C. Morris, and P. Mutzel, "Classifying dissemination processes in temporal graphs," *Big Data*, vol. 8, no. 5, pp. 363–378, 2020.

[22] X. Meng, W. Li, X. Peng, Y. Li, and M. Li, "Protein interaction networks: centrality, modularity, dynamics, and applications," *Frontiers of Computer Science*, vol. 15, pp. 1–17, 2021.

[23] L. Paulevé, J. Kolčák, T. Chatain, and S. Haar, "Reconciling qualitative, abstract, and scalable modeling of biological networks," *Nature communications*, vol. 11, no. 1, p. 4256, 2020.

[24] J. Barunik, M. Ellington *et al.*, "Dynamic networks in large financial and economic systems," *arXiv preprint arXiv:2007.07842*, 2020.

[25] N. Nonejad, "An overview of dynamic model averaging techniques in time-series econometrics," *Journal of Economic Surveys*, vol. 35, no. 2, pp. 566–614, 2021.

[26] X. Tannier and P. Muller, "Evaluating temporal graphs built from texts via transitive reduction," *Journal of Artificial Intelligence Research*, vol. 40, pp. 375–413, 2011.

[27] J. Saramäki, M. Kivelä, and M. Karsai, "Weighted temporal event graphs," *Temporal Network Theory*, pp. 107–128, 2019.

[28] G. Guidi, O. Selvitopi, M. Ellis, L. Oliker, K. Yelick, and A. Buluç, "Parallel string graph construction and transitive reduction for de novo genome assembly," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 517–526.

[29] B. B. Xuan, A. Ferreira, and A. Jarry, "Computing shortest, fastest, and foremost journeys in dynamic networks," *International Journal of Foundations of Computer Science*, vol. 14, no. 02, pp. 267–285, 2003.

[30] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, "Temporal distance metrics for social network analysis," in *Proceedings of the 2nd ACM workshop on Online social networks*, 2009, pp. 31–36.

[31] V. Kostakos, "Temporal graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[32] O. Michail, "An introduction to temporal graphs: An algorithmic perspective," *Internet Mathematics*, vol. 12, no. 4, pp. 239–280, 2016.

[33] L. Oettershagen, "Temporal graph algorithms," Ph.D. dissertation, Universitäts-und Landesbibliothek Bonn, 2022.

[34] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial intelligence and statistics*. PMLR, 2009, pp. 488–495.

[35] T. H. Schulz, T. Horváth, P. Welke, and S. Wrobel, "A generalized weisfeiler-lehman graph kernel," *Machine Learning*, vol. 111, no. 7, pp. 2601–2629, 2022.

[36] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *arXiv preprint arXiv:2007.08663*, 2020.

[37] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.

[38] A. A. Hagberg, D. Schult, and P. Swart, "Networkx," https://networkx.github.io/, 2008–, accessed: May 8, 2023.

[39] P. Procházka, M. Mareš, and M. Dědič, "Scalable graph size reduction for efficient gnn application," 2022.