UNIVERSITÀ DEGLI STUDI DI SIENA

Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche

UNIVERSITÀ
DI SIENA
1240

Temporal Graph Processing and Pooling in Graph Neural Networks

Veronica Lachi

PhD in Information Engineering and Science

Supervisors

Prof. Monica Bianchini, Prof. Chiara Mocenni

Examination Commitee

Prof. Fabrizio Costa

Prof. Barbara Hammer

Prof. Stefano Melacci

Thesis reviewers

Prof. Fabrizio Costa

Prof. Josephine Maria Thomas

Siena, 16/07/2024

Ringraziamenti

Desidero esprimere i miei più profondi ringraziamenti alla mia Advisor, Prof.ssa Monica Bianchini, per avermi permesso di intraprendere questo percorso, per avermi seguita e supportata costantemente e per avermi concesso la libertà di esplorare gli argomenti di ricerca di mio interesse. La ringrazio inoltre per aver sempre promosso un ambiente di lavoro sereno e disteso, privo di competizioni o stress. Un sincero ringraziamento va anche alla mia Co-Advisor, Prof.ssa Chiara Mocenni, per la guida, i preziosi consigli e il costante supporto. Ringrazio di cuore il Prof. Franco Scarselli che, pur non essendo ufficialmente mio advisor, mi ha dedicato molto tempo ed energia, seguendomi con pazienza e trasmettendomi la passione per le sue GNN. Un sentito ringraziamento va al Prof. Filippo Maria Bianchi, che durante il mio periodo di visita all'UiT, e anche successivamente, è stato una guida fondamentale per il mio percorso. Lo ringrazio per avermi seguita scrupolosamente, per avermi incoraggiata ed aiutata a credere maggiormente in me stessa, e per tutti i consigli sul mondo della ricerca che mi ha dispensato.

I primi e più grandi ringraziamenti non istituzionali vanno a Caterina, con la quale ho condiviso ogni momento ed ogni emozione di questo percorso. La ringrazio per avermi insegnato come studiare e fare la matematica, la ringrazio per tutto l'inchiostro che abbiamo versato insieme per scrivere definizioni e dimostrare teoremi, la ringrazio per la sincerità e la serietà con cui mi ha consigliato in ogni lavoro. Ma questo è solo una parte infinitesimale di quello che ci lega. Non ho contezza di quanto bene mi abbiano fatto al cuore le risate, le chiacchiere sul futuro, i viaggi in macchina, le esperienze in giro per il mondo. Non avrei mai pensato che lo studio potesse farmi il dono così grande di incontrare una persona così simile a me e un'amica così speciale. Non la ringrazierò mai abbastanza per aver occupato nel mio cuore uno spazio così importante.

Grazie a tutti i colleghi e amici del Lab202, attori protagonisti di questo bellissimo e divertentissimo film durato tre anni. Sono certa non esista al mondo un laboratorio di ricercatori altrettanto belli e simpatici. Gli anni passati con loro sono stati anni di leggerezza e gioia. Sono grata di essere capitata in mezzo a loro. Un grazie speciale a Paolo che per primo mi ha supportata ad inizio dottorato, quando era tutto difficile, e che da allora è una presenza costante nella mia vita. Grazie anche ad Alessio che mi ha fatto passare al lato oscuro della teoria delle GNN e con cui ho trascorso dei momenti indimenticabili.

Un grande grazie ai colleghi e amici incontrati durante il periodo di ricerca a Tromsø. In particolare, grazie a Michele, Giulia, Ettore e Robin con i quali ho condiviso l'esperienza più bella

della mia vita in Norvegia, il mio sogno da sempre, e che anche adesso a 3929 km di distanza sono sempre presenti.
Un sentito ringraziamento alle mie colleghe da Kassel, Alice e Silvia. Grazie ad Alice per essere la mia pazza sorella tedesca e a Silvia per la sua dolcezza e per le sue focacce.
Vorrei ringraziare tutte le persone a me care fuori dal mondo accademico che, nonostante in questi anni abbia passato la maggioranza del tempo china su paper o davanti al mio computer, sono pazientemente rimaste al mio fianco.
Ringrazio le mie amiche del liceo Arianna, Ilaria e Laura, che, nonostante credessero studiassi Statistics in Economics, mi hanno sempre supportato e consigliato con curiosità e amore, e che sono sempre state con me, oltre qualsiasi distanza.
Grazie alle mie amiche di sempre Eleonora, Giulia, Maira, Silvia, Sofia e Rebecca per il loro costante sostegno e per avermi regalato i migliori momenti di spensieratezza.
Grazie a Sara, che nonostante gli impegni, il lavoro e la distanza ha sempre trovato tempo ed energie da dedicarmi e che è diventata in breve tempo un'amica insostituibile.
La mia più profonda gratitudine va alla mia famiglia, il punto saldo di tutta la mia vita. Grazie alla mamma e al babbo per avermi dato i mezzi morali e materiali per arrivare al gradino più alto degli studi. Grazie per tutto l'impegno che hanno messo nel cercare di avvicinarsi a questo mondo così distante dal loro. Grazie di cuore alla mia Tata, che è stata la prima ad iniziarmi allo studio insegnandomi a leggere e a scrivere. Grazie per essere da sempre un modello per me, spronandomi a puntare sempre più in alto, e grazie per essere la persona che meglio mi conosce al mondo. Grazie alla mia Livia, che mi ha reso zia e che mi ricorda ogni giorno l'importanza di essere curiosi.
Il dottorato è stato il periodo più bello della mia vita. Ho viaggiato, vissuto lontano da casa e conosciuto tanto altro rispetto a quello che era in me. Spero che questa non sia una fine, ma l'inizio di un lungo percorso per aprire sempre di più la mia mente e conoscere me stessa e il mondo.

Abstract

Graph Neural Networks (GNNs) have emerged as a superior technique for handling graph-based data, outperforming traditional methods in a multitude of domains. Indeed, the unique, non-Euclidean structure of graph data introduces specific challenges that complicate the use of standard neural network methodologies. This thesis tackles two important issues: the integration of temporal dynamics into graph structures and the reduction of dimensionality necessary for developing truly deep neural architectures. The first part of this thesis presents a thorough review and taxonomy of existing Graph Neural Network models that handle temporal graphs, detailing their applications in various real-world scenarios. Additionally, it presents a universal approximation theorem for a specific class of temporal GNNs. The second part delves into strategies for graph dimensionality reduction via hierarchical pooling methods. This exploration includes an analysis of how these pooling operators affect the overall expressive power of GNNs. In particular, sufficient conditions for the pooling operator to maintain and even enhance the expressive power of GNNs are presented. By addressing these advanced concepts, the thesis aims to deepen the understanding of GNN optimization for dynamic and complex graph-based applications, setting a foundation for future research and application improvements in the field.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 3
- 1.2 Outline . . . 4
- 1.3 List of Publications . . . 4

2 Preliminaries **7**
- 2.1 Graph Neural Networks . . . 7
  - 2.1.1 WL test . . . 8
  - 2.1.2 Expressive Power of Graph Neural Networks . . . 9

3 Temporal Graph Neural Networks **11**
- 3.1 Temporal Graphs: Basic Concepts and Definitions . . . 11
  - 3.1.1 Representation of temporal graphs . . . 12
- 3.2 Learning Settings . . . 13
- 3.3 Supervised Learning Tasks . . . 15
  - 3.3.1 Classification . . . 15
  - 3.3.2 Regression . . . 16
  - 3.3.3 Link Prediction . . . 17
- 3.4 Unsupervised Learning Tasks . . . 17
  - 3.4.1 Clustering . . . 17
  - 3.4.2 Anomaly detection . . . 18
  - 3.4.3 Low-dimensional embedding (LDE) . . . 19
- 3.5 Taxonomy of Temporal Graph Neural Networks . . . 19
  - 3.5.1 Snapshot-based models . . . 20
  - 3.5.2 Event-based models . . . 22
  - 3.5.3 Category comparison . . . 24
- 3.6 Open Challenges . . . 24
- 3.7 The Expressive Power of Temporal Graph Neural Networks . . . 26
  - 3.7.1 Dynamic WL test . . . 26
  - 3.7.2 Universal Approximation Theorem for Temporal Graph Neural Networks 28
  - 3.7.3 Experimental Results . . . 29

3.8 Other approaches to model temporal graphs 32
3.9 Conclusions 33

4 The Expressive Power of Pooling in Graph Neural Networks 35
4.1 Pooling in Graph Neural Networks: Basic Concepts and Definitions 36
4.1.1 Select, Reduce, Connect 37
4.1.2 Taxonomy of Graph Pooling 37
4.1.3 Existing Pooling Operators 39
4.1.4 Evaluation of a Pooling Operator 42
4.2 Pooling Operators can Preserve the Expressive Power of GNNs 42
4.2.1 Conditions for Preserving the Expressive Power 43
4.2.2 Expressiveness of Existing Pooling Operators 45
4.2.3 Criticism on Pooling 48
4.2.4 Experimental Results 48
4.3 Pooling Operators can Increase the Expressive Power of GNNs 54
4.3.1 Conditions for Increasing the Expressive Power 56
4.3.2 XPool: An Expressive Pooling Operator 58
4.3.3 Experimental Results 59
4.4 Conclusions 61

5 Conclusion 63

A Appendix 65

Bibliography 71

List of Figures

2.1 Two non-isomorphic but WL equivalent graphs. 9

3.1 **Learning settings.** Schematic representation of the learning settings on TGs formalized in Section 3.2. The temporal graphs are represented as sequences of snapshots, with training (red) and inference (green) nodes connected by edges (solid lines), and where a dotted line connects instances of the same node (with possibly different features and/or labels) in successive snapshots. The four categories are obtained from the different combinations of a temporal and a topological dimension. The **temporal** dimension distinguishes the *future* setting, where the training nodes are all observed before the inference nodes (first row), from the *past* setting where inference is performed also on nodes appearing before the observation of the last training node (second row). The **topological** dimension comprises a *transductive* setting, where each inference node is observed (unlabelled) also during training (left column), and an *inductive* setting, where inference is performed on nodes that are unknown at training time (right column). 14

3.2 **The proposed TGNN taxonomy with an analysis of the surveyed methods.** The top panel shows the categories with the corresponding model instances (Section 3.5), where the colored bullets additionally indicate the main technology that they employ. The bottom table maps these methods to the task (Sections 3.3,3.4) to which they have been applied in the respective original paper, with an additional indication of their use in the future (F), past (P), inductive (I), or transductive (T) settings (Section 3.2). Note that no method has been applied yet to clustering and visualization, for neither graphs nor nodes. Moreover, only four out of ten models have been tested in the past mode (three in PT, one in PI). 20

3.3 Two STGs that are WL equivalent in each snapshots but DWL not equivalent. . 27

3.4 The four static graphs are used as components to generate the synthetic dataset. Graphs a) and b) are equivalent under the static WL test; the same holds for c) and d). 30

3.5 Experimental Framework **E1.** Training accuracy over the epochs for an STGNN trained on the dataset containing STGs up to time length $T = 4$ (a) and $T = 5$ (b). 31

3.6 Experimental framework **E2.** Training accuracy a) and training loss b) over the epochs for several STGNNs trained on the dataset containing STGs up to time length $T = 5$. Figure b) is in logarithmic scale. 31

4.1 A schematic representation of a graph pooling operator. 36
4.2 A schematic representation of the SRC framework. Adapted from [1]. 37
4.3 Taxonomy of pooling operators based on the SRC framework. Adapted from [1]. 39
4.4 A flat GNN is at most as powerful as the WL test, while no results have been proven so far for hierarchical GNNs. 43
4.5 A GNN with expressive MP layers (condition 1) computes different features for two graphs G_1, G_2 that are WL-distinguishable. A pooling layer satisfying the conditions 2 and 3 generates coarsened graphs G_{1_P} and G_{2_P} that are still WL-distinguishable. 45
4.6 Example of failure of Top-k pooling. Given two WL-distinguishable graphs with node features $x_1 \leq x_2 \leq x_3 \leq x_4$, two scoring vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ are computed using a projector p. Then, the two nodes associated with the highest scores are selected. If $\boldsymbol{p \leq 0}$, nodes **1** and **2** are chosen in both graphs. Conversely, if $\boldsymbol{p > 0}$, nodes **3** and **4** are selected. Therefore, regardless of the value learned for the projector p, the two input graphs will be mapped into the same pooled graph. . . 47
4.7 Three pairs of graphs from the EXPWL1 dataset. Each pair consists of two graphs with different classes that are WL distinguishable. 50
4.8 Average accuracy (and std.) v.s. average runtime on the benchmark datasets. . . 53
4.9 Two WL-indistinguishable graphs G_1, G_2 which can be distinguished after using a powerful `SEL`. Clustering cycles maps G_1 to a two supernode graph, while it maps G_2 to a single supernode graph. 57
4.10 Two WL-indistinguishable graphs G_1, G_2 which can be distinguished after pooling. Contracting edges and adding a single superedge between supernodes iff any original nodes were connected results in a disconnected graph for G_1 and a connected graph for G_2. 58

A.1 The ATTACH operator on trees. 70

List of Tables

4.1 Performance of baseline architectures on EXPWL1. 49
4.2 Details of the graph classification datasets. 51
4.3 Classification results on EXPWL1. 52
4.4 Graph classification test accuracy on benchmark datasets. 54
4.5 Graph classification test run-time in s/epoch. 55
4.6 Number of graph pairs correctly distinguished by each GNN using different pooling operators. Mean and standard deviation over three runs are shown. 61

Chapter 1

Introduction

Recent advances in Deep Learning techniques, particularly through the use of deep neural networks, have made them applicable to a wide range of complex and varied tasks. These models have shown exceptional performance in processing speech [2, 3], images [4, 5, 6, 7], and videos [8], which all share a Euclidean structure.

However, there is an increasing interest in exploring these techniques on non-Euclidean data, such as graphs. Graphs are data structures composed of collections of nodes and edges, which can be used to represent objects, or patterns, along with their relationships. This type of data is prevalent in several application domains, including social network [9, 10], sensor network [11] analysis, genetic research [12] and so on.

The inherent non-Euclidean nature of graphs presents unique challenges, such as the absence of global parameterization and shift invariance, making operations like convolution, common in Euclidean spaces, not directly applicable. Despite these challenges, the complexity of graph structures, especially evident in large-scale networks, makes them ideal candidates for machine learning applications.

Graph Neural Networks (GNNs) [13] have emerged as powerful tools for handling graph-structured data, enabling direct processing of relational information and facilitating computations on nodes or edges with minimal information loss. GNNs conceptualize the input as entities and their relationships, evolving through innovations in model architecture, towards Graph Convolution Networks (GCNs) [14], GraphSAGE [15], and Graph Attention Networks (GATs) [16]. These models have been used effectively in various fields, from computer vision [17], recommendation systems [18] and software engineering [19, 20], up to biomedicine [21, 22] and new drug discovery [23, 24].

GNNs address three primary types of tasks: node-focused, edge-focused, and graph-focused challenges. In node-focused tasks, the model outputs a value for each node or a subset of nodes for purposes like classification or clustering. Edge-focused tasks involve understanding relationships between nodes, predicting connections, or classifying patterns. Graph-focused tasks aim at deducing properties or clustering the entire graph representation. The versatility and adaptability of GNNs make them highly effective for a wide spectrum of applications, whether it be regression or classification [25], both in inductive and transductive learning contexts [26, 27].

GNNs have been extensively studied not only for their practical applications but also for their theoretical properties, including approximation and generalization capabilities as well as their expressive power. In [28], a landmark study on the expressive power of GNNs is presented, leveraging the concept of the Weisfeiler-Lehman (WL) test for graph isomorphism as a benchmark for GNNs' ability to distinguish graph structures. This study emphasizes that, while GNNs are highly expressive, their power is fundamentally limited to that of the WL test. On the other

hand, [29] suggests that, with sufficient depth and width, GNNs can approximate any function that is invariant or equivariant to graph isomorphism. Regarding generalization, [30] introduced generalization bounds on VC dimension. Finally, in [31], further bounds VC dimension are derived and related to the number of colors produced by the WL test.

As far as now, most of the theoretical studies have been dedicated to the general case, where a basic GNN is employed and simple static graphs are processed. On the other hand, research on GNNs has rapidly evolved giving rise to models often specialized in specific tasks and graph domains. This thesis focuses on two classes of models that are particularly relevant for current applications, namely the GNN models for dynamic graphs and large graphs.

In fact, it is essential to acknowledge that, in real-world applications, graph-structured data exhibits a temporally varying structure. Indeed, the connections between entities change over time, with some entities appearing or disappearing and the attributes associated with entities also undergoing changes. The ability to process temporal graphs is becoming increasingly important in a variety of fields such as recommendation systems [32, 33], social network analysis [34, 35], transportation systems [36, 37, 38], modeling of face-to-face interactions [39], human mobility [40, 41, 42], epidemic modeling and contact tracing [43, 44], and many others. Traditional graph-based models are not well suited for analyzing temporal graphs as they assume a fixed structure and are unable to capture its temporal evolution.

Therefore, in the last few years, several models capable to directly encode temporal graphs have been developed, such as matrix factorization-based approaches [45] and temporal motif-based methods [46]. Recently, also GNNs have been successfully applied to temporal graphs. Indeed, their success in various static graph tasks, including node classification [15, 47, 14, 48, 49, 50, 51] and link prediction [52, 53, 54], has not only established them as the leading paradigm in static graph processing, but has also indicated the importance of exploring their potential in other graph domains, such as temporal graphs. With approaches ranging from attention-based methods [55] to Variational Graph-Autoencoders (VGAEs) [56], Temporal Graph Neural Networks (TGNNs) have achieved state-of-the-art results on tasks such as temporal link prediction [57], node classification [58] and edge classification [59].

The first objective of this thesis is to provide a systematization of existing GNN-based methods for temporal graphs and a formalization of the tasks addressed. In particular, we provide a comprehensive overview of learning settings and tasks that can be performed on temporal graphs, classify existing work on TGNNs into a detailed taxonomy, discuss limitations of current TGNN methods, propose new research challenges, and identify potential ad high impact applications for TGNNs. Finally, we present new versions of the WL test appropriate for temporal graphs and we show that some TGNNs are capable of approximating, in probability and up to any precision, any measurable function on graphs that respects the WL equivalence. We further validate our theoretical findings through empirical experiments.

Similarly to temporal graphs, large graphs are widely diffused in modern applications and require specialized GNN models. The second part of the thesis focuses on graph dimensionality reduction through pooling, which is a main mechanism by which GNNs face large graph. Actually, reducing input data size has been a staple in machine learning to cut down on complexity and to cluster vital information effectively. Inspired by the conventional pooling methods used in Convolutional Neural Networks (CNNs) [60, 61], pooling has been adapted for GNNs, especially for graph-level tasks.

However, the transition from image to graph data introduces significant challenges. Unlike images, which have a regular, tabular structure allowing for straightforward dimensionality reduction through techniques like mean or max pooling, graphs represent a more complex scenario. Their irregular structure complicates the process, demanding more sophisticated approaches to reduce dimensions without losing essential information.

Addressing this challenge, extensive research has delved into developing various pooling methods, evolving from basic strategies that consider just the topolgy of the graph, like Graclus [62], to more sophisticated models that can be learned end-to-end along with the other components of the GNN architecture, like MinCut [63] and DiffPool [64]. By creating layers of increasingly abstract representations, pooling allows GNNs to capture both local and global graph structures. GNNs equipped with adequate pooling layers have been shown to outperform those without them, highlighting the critical role of effective dimensionality reduction [63].

The importance of pooling in GNNs underscores the need for ongoing investigation into its theoretical properties. As such, we present some findings related to the relationship between pooling and the expressive power of GNNs. While all the studies about expressiveness of GNNs have focused on flat GNNs, which consist solely of sequential message-passing (MP) layers, our discussion extends to hierarchical GNNs, i.e., composed by MP layers interleaved with pooling ones. We examine critical research questions, such as the conditions under which pooling can be implemented without decreasing the GNNs' expressive power and whether it is feasible to design pooling layers that not only preserve but enhance GNNs' expressiveness.

1.1 Contributions

The main contributions of this thesis are the following. The letter placed at the end of each contribution refers to the article in which that contribution was proposed. A list of these articles, along with their corresponding letters, can be found at the end of this chapter.

- We propose a coherent formalization of the different learning settings and of the tasks that can be performed on temporal graphs. Thus, we unify existing formalism and informal definitions that are scattered in the literature, and highlight substantial gaps in what is currently being tackled; (D)

- We organize existing TGNN works into a comprehensive taxonomy that groups methods according to the way in which time is represented and the mechanism with which it is taken into account; (D)

- We highlight the limitations of current TGNN methods, discuss open challenges that deserve further investigation and present critical real-world applications where TGNNs could provide substantial gains; (D)

- We present a new version of the WL test for temporal graphs (DWL) and we show that a specific class of TGNN models is capable of approximating, in probability and up to any precision, any measurable function on temporal graphs that respects the DWL equivalence; the proof is based on space partitioning, which allows us to deduce information about the TGNN architecture that can achieve the desired approximation; (B)

- We show that, when certain conditions are met in the MP layers and in the pooling operator, their combination produces an injective function between graphs. This implies that the GNN can effectively coarsen the graph to learn high-level data descriptors, without compromising its expressive power; (A)
- Based on our theoretical analysis, we identify commonly used pooling operators that do not satisfy these conditions and may lead to failures in certain scenarios;
- We introduce a simple yet effective experimental setup for measuring, empirically, the expressive power of any GNN in terms of its capability to perform a graph isomorphism test; (A)
- We define sufficient conditions for a pooling operator to increase expressiveness; (C)
- We develop a new hierarchical pooling method called XPool, specifically designed to satisfy conditions sufficient to enhance the expressiveness of the GNN in which it is integrated; experiments on synthetic datasets, purposely crafted to test GNN expressiveness, show that XPool increases the expressiveness of a GNN, even when it includes not-powerful message passing layers. (C)

1.2 Outline

The thesis is organized as follows. In Chapter 2, we lay the groundwork for the understanding of the rest of the manuscript, by providing essential preliminary definitions. This includes the notation used throughout the thesis and the foundational concepts necessary for its comprehension. Chapter 3 shifts the focus to the analysis of models based on GNNs for the study of temporal graphs. After defining the fundamental concepts specific to temporal graphs, we present a taxonomy of a class of these models, followed by an exploration of their expressiveness. In Chapter 4, we delve into the concept of pooling within GNNs; we start by defining what pooling entails in the context of GNNs, then review the most significant pooling models identified in the literature, and conclude with theoretical and empirical findings on the expressiveness of GNNs incorporating pooling operations. Finally, in Chapter 4, some conclusions are drawn, and further developments are suggested.

1.3 List of Publications

My doctoral program has brought to the publication of the following papers, inherent to this thesis.

A. Filippo Maria Bianchi and **Veronica Lachi**, *The Expressive Power of Pooling in Graph Neural Networks*, Advances in Neural Information Processing Systems 36 (2024), [65];

B. Silvia Beddar-Wiesing, Giuseppe Alessio D'Inverno, Caterina Graziani, **Veronica Lachi**, Alice Moallemy-Oureh, Franco Scarselli and Josephine Maria Thomas, *Weisfeiler–Lehman goes dynamic: An analysis of the expressive power of Graph Neural Networks for attributed and dynamic graphs*, Neural Networks (2024), [66];

C. **Veronica Lachi**, Alice Moallemy-Oureh, Andreas Roth and Pascal Welke, *Graph Pooling Provably Improves Expressivity*, NeurIPS 2023 Workshop: New Frontiers in Graph Learning (2023), [67];

D. Antonio Longa, **Veronica Lachi**, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Liò, Franco Scarselli and Andrea Passerini, *Graph Neural Networks for Temporal Graphs: State of the Art, Open Challenges, and Opportunities*, Transactions on Machine Learning Research (2023), [68].

The program was instrumental to study other topics, which are not included to include in this thesis, and have produced the following publications:

- **Veronica Lachi**, Francesco Ferrini, Antonio Longa, Bruno Lepri and Andrea Passerini, *A Simple and Expressive Graph Neural Network Based Method for Structural Link Representation*, ICML 2024 GRaM Workshop, [69].
- Pietro Bongini, Niccolò Pancino, **Veronica Lachi**, Caterina Graziani, Giorgia Giacomini, Paolo Andreini and Monica Bianchini, *Point-Wise Ribosome Translation Speed Prediction with Recurrent Neural Networks*, Mathematics (2024), [22];
- Silvia Beddar-Wiesing, Giuseppe Alessio D'Inverno, Caterina Graziani, **Veronica Lachi** and Alice Moallemy-Oureh, *On the Extension of the Weisfeiler-Lehman Hierarchy by WL Tests for Arbitrary Graphs*, 18th International Workshop on Mining and Learning with Graphs (2022), [70];
- Paolo Andreini, Simone Bonechi, Giorgio Ciano, Caterina Graziani, **Veronica Lachi**, Natalia Nikoloulopoulou, Monica Bianchini and Franco Scarselli, *Multi-stage Synthetic Image Generation for the Semantic Segmentation of Medical Images*, Artificial Intelligence and Machine Learning for Healthcare (2022), [71];
- Giorgia Giacomini, Caterina Graziani, **Veronica Lachi**, Pietro Bongini, Niccolò Pancino, Monica Bianchini, Davide Chiarugi, Angelo Valleriani and Paolo Andreini, *A neural network approach for the analysis of reproducible Ribo–Seq profiles*, Mathematics (2022), [21];
- Paolo Andreini, Giorgio Ciano, Simone Bonechi, Giuseppe Alessio D'Inverno, Cateri-na Graziani, **Veronica Lachi**, Alessandro Mecocci, Andrea Sodi, Franco Scarselli and Monica Bianchini, *A two-stage GAN for high-resolution retinal image generation and segmentation*, Electronics, (2021), [6];
- Niccolò Pancino, Caterina Graziani, **Veronica Lachi**, Maria Lucia Sampoli, Emanuel Ștefănescu, Monica Bianchini and Giovanna Maria Dimitri, *A mixed statistical and machine learning approach for the analysis of multimodal trail making test data*, Mathematics (2021), [7].

Chapter 2

Preliminaries

This chapter introduces and defines the basic concepts that are crucial for the rest of the thesis. We start with the definition of graphs; next, we introduce Graph Neural Networks, the primary machine learning model used in this work for handling data structured as graphs. Lastly, we will provide a brief overview of the literature concerning the expressiveness of GNNs. This includes discussing the capabilities and limitations of GNNs as highlighted in existing research. This foundation will support the novel contributions and discussions presented in later chapters.

2.1 Graph Neural Networks

Definition 2.1.1 (Graph)**.** *A graph is a tuple $G = (V, E)$, where V is the set of nodes with $|V| = N$ and $E \subseteq V \times V$ is the set of edges with $|E| = M$. To each graph G is associated a matrix of node features, $\mathbf{X} \in \mathbb{R}^{N \times F}$, where the v-th row is the feature vector $\mathbf{x}_v \in \mathbb{R}^F$ of node v. Analogously, $\mathbf{X}^E \in \mathbb{R}^{M \times F}$ is the edge feature matrix. The adjacency matrix $A \in \mathbb{R}^{N \times N}$ is such that $A_{uv} = 1$, if $(u, v) \in E$, and $A_{ij} = 0$, otherwise.*

We assume that all graphs are directed, i.e., $(u, v) \in E$ does not imply that $(v, u) \in E$. Moreover, given $v \in V$, the set

$$\mathcal{N}[v] := \{u \in V : (u, v) \in E\}$$

denotes the neighborhood of v in G.
In the remainder of the document, we will use the notation $\{\cdot\}$ to represent a multiset. A multiset is a generalization of the concept of set, allowing for multiple instances of its elements. Unlike a traditional set, where each element is unique and appears only once, a multiset can contain several occurrences of the same element.

GNNs have emerged as a powerful class of neural network models specifically designed to tackle data structured in the form of graphs. The first theorization of GNNs dates back to 2005 [72] — with the full mathematical formulation proposed in 2009 [13] — and describes them as networks which replicate the topology of the input graph, and exchange messages between neighbor nodes to produce an output on selected nodes (or edges or for the whole graph).

GNNs rely on the so called message passing (MP) mechanism, which implements a local computational scheme to process graphs. Formally, the information related to a node v is stored into a feature vector $\mathbf{x}_v^l$ that is iteratively updated by aggregating the features of its neighboring nodes. The representation of each node is initialized with its feature, i.e., $\mathbf{x}_v^0 := \mathbf{x}_v$ and after l iterations, the vector $\mathbf{x}_v^l$ contains both the structural information and the node content of the

l–hop neighborhood of v. Given a sufficient number of iterations, the node feature vectors can be used to classify the nodes or the entire graph. Specifically, the output of the l-th layer of a message passing GNN is:

$$\mathbf{x}_v^l = \texttt{COMBINE}^{(l)}(\mathbf{x}_v^{l-1}, \texttt{AGGREGATE}^{(l)}(\{\mathbf{x}_u^{l-1},\ u \in \mathcal{N}[v]\})) \tag{2.1}$$

where $\texttt{AGGREGATE}^{(l)}$ is a function that aggregates the node features from the neighborhood $\mathcal{N}[v]$ at the $(l-1)$-th iteration, and $\texttt{COMBINE}^{(l)}$ is a function that combines the features of the node v with those of its neighbors. The aggregation step often involves employing permutation invariant operations such as mean, max-pooling, and sum. These operations ensure that the final aggregated representation is insensitive to the ordering of the nodes. Instead, typical choices for the `COMBINE` function are concatenation and summation. In node level tasks, a `READOUT` function is used to produce an output for each node, based on its features at the final layer L:

$$\mathbf{h}_v \ = \ \texttt{READOUT}(\mathbf{x}_v^L)$$

whereas, in graph level tasks, a `READOUT` function produces the final output given the feature vectors from the last layer L:

$$\mathbf{h} \ = \ \texttt{READOUT}(\{\mathbf{x}_v^L,\ v \in V\}).$$

Based on the various implementations of the `COMBINE` and `AGGREGATE` functions, several distinct GNN models have been developed, including GCN [14], GAT [47], and GIN [28]. In the following, we will explore models that specifically tackle temporal graphs employing this kind of message passing scheme, which we refer to as Temporal Graph Neural Networks (TGNNs).

When evaluating the expressive power of GNNs, the focus is on their ability to produce distinct outputs for different graphs or nodes. In this context, the study of GNNs' expressive power is intimately linked to the graph isomorphism problem, which assesses whether two graphs are structurally identical. Ideally, GNNs should be able to differentiate between non-isomorphic graphs by producing different outputs. However, graph isomorphism presents a significant challenge, as it is a complex problem for which no polynomial-time algorithms have yet been identified. The Weisfeiler-Lehman (WL) [73] tests stand out as a series of effective and computationally feasible solutions for distinguishing between a wide class of graphs. Particularly, the WL test, resembling the process of neighborhood aggregation in GNNs, offers a theoretical foundation for understanding and improving GNNs' capabilities in graph structure differentiation.

2.1.1 WL test

The WL test is a method that tests whether two graphs are isomorphic, based on a graph coloring refinement procedure. The WL test iteratively updates colors assigned to each node based on the colors of its neighboring nodes. The coloring algorithm is applied in parallel on the two input graphs. At the end, the multisets of node colors of the two graphs are compared: if they are equal, then the graphs are possibly isomorphic, whereas if they are not equal, then the graphs are certainly non-isomorphic. Note that the test is not conclusive in the case of a positive response, as the graphs may still be non-isomorphic. Node colors are initialized based on the node features and then updated using the previous iteration coloring.

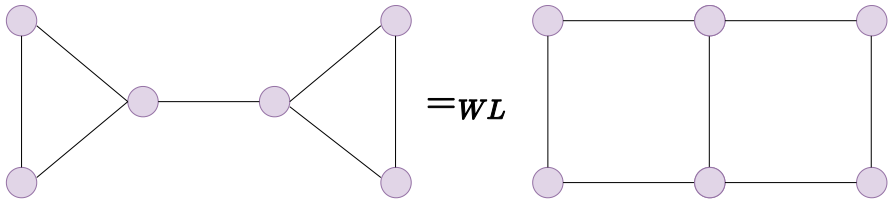


Figure 2.1: Two non-isomorphic but WL equivalent graphs.

1. At iteration 0, the node colors are initialized as:

$$c_v^0 = \texttt{HASH}^0(\mathbf{x}_v)$$

where $\texttt{HASH}^0$ is a function that injectively codes every possible feature with a color.

2. For any iteration $l > 0$, the color is refined in the following way:

$$c_v^l = \texttt{HASH}(c_v^{l-1}, \{c_u^{l-1} | u \in \mathcal{N}[v]\})$$

where HASH is an injective map.

The algorithm terminates when the multiset of colors does not change between two successive iterations.

2.1.2 Expressive Power of Graph Neural Networks

There is a strong analogy between an iteration of the WL-test and the aggregation scheme implemented by message passing in GNNs. In fact, it has been proven that GNNs are at most as powerful as the WL test in distinguishing different graph-structured data [28, 74]. Moreover, if the message passing operation is injective, the resulting GNN is as powerful as the WL test [28]. The Graph Isomorphism Network (GIN) implements such an injective multiset function as:

$$\mathbf{x}_v^l = \texttt{MLP}^{(l)}\left((1+\epsilon^l)\mathbf{x}_v^{l-1} + \sum_{u \in \mathcal{N}[v]} \mathbf{x}_u^{l-1}\right). \tag{2.2}$$

Under the condition that the nodes' features are from a countable multiset, the representational power of GIN equals that of the WL test.
However, the WL test, a heuristic for graph isomorphism, has known limitations in distinguishing certain graph structures, meaning cases where the graphs are non-isomorphic but the test produces the same multiset of colors as their output. Consequently, GNNs will also produce equal outputs on these graphs. An example of such a failure is presented in Figure 2.1.

This poses a significant challenge in terms of expressiveness, particularly in domains demanding high-precision recognition of complex patterns, such as chemistry. The WL test's inability to differentiate between molecules with similar but non-identical structures can lead to inaccuracies in predicting molecular properties, which is detrimental to drug discovery and materials science where such predictions play a foundational role.
In response to these limitations, researchers have proposed several strategies to enhance the

expressiveness of GNNs beyond that of the WL test. One approach involves incorporating higher-order interactions among graph elements, enabling the network to capture more complex structural details [74]. In alternative, it is possible to increase the expressiveness of GNNs by aggregating multisets of subgraphs [75] or path of nodes [76] instead of multisets of node features. Another strategy is the augmentation of node features with structural roles or positions within the graph, providing a richer context for each node [77]. However, these approaches can lead to high computational cost or generalization problems. In chapter 4, we will present a new method for enhancing the expressive power of GNNs reducing the complexity and amount of computations and maintaining the standard message passing scheme, through the use of a suitable hierarchical pooling layer.

Chapter 3

Temporal Graph Neural Networks

Traditionally, machine learning models for graphs have been mostly designed for static graphs [78, 79]. However, many applications involve temporal graphs [80, 81, 82]. This introduces important challenges for learning and inference since nodes, attributes, and edges change over time.

This chapter delves into the analysis of Temporal Graph Neural Networks (TGNNs), a field that extends traditional graph neural networks to meet the dimension of time. The main presented results are based on the papers [68, 66].

Our discussion begins by providing fundamental definitions that lay the groundwork for understanding temporal graphs. Following the foundational groundwork, we will focus on strategies that leverage the message passing scheme presented in 2.1 for dealing with temporal graphs, namely TGNNs. A central section of this chapter is dedicated to a theoretical analysis of the expressive power of TGNNs. We will analyse the capabilities and limitations of TGNNs, providing insights into how they model temporal relationships and what makes them uniquely suited for time-sensitive data.

By weaving together fundamental concepts, advanced methodologies, theoretical insights, and practical applications, this chapter aims to provide a comprehensive overview of temporal graph processing through TGNNs.

3.1 Temporal Graphs: Basic Concepts and Definitions

In this section, a formal definition of temporal graphs is provided by structuring different existing notions in a common framework.

Definition 3.1.1 (Temporal Graph - TG)**.** *A* Temporal Graph *is a tuple* $G_T = (V, E, V_T, E_T)$*, where* V *and* E *are, respectively, the set of all possible nodes and edges appearing in* G_T *at any time, while*

$$V_T := \{(v, \mathbf{x}_v, t_s, t_e) : v \in V, \mathbf{x}_v \in \mathbb{R}^F, t_s \leq t_e\},$$
$$E_T := \{(e, \mathbf{x}_e, t_s, t_e) : e \in E, \mathbf{x}_e \in \mathbb{R}^F, t_s \leq t_e\},$$

are the temporal nodes and edges, with time-dependent features and initial and final timestamps. A set of temporal graphs is denoted as $\mathcal{G}_\mathcal{T}$*.*

Observe that we implicitly assume that the existence of a temporal edge in E_T requires the simultaneous existence of the corresponding temporal nodes in V_T. Moreover, the definition implies that node and edge features are constant inside each interval $[t_s, t_e]$, but may otherwise change

over time. Since the same node or edge may be listed multiple times, with different timestamps, we denote with $\bar{t}_s(v) = \min\{t_s : (v, \mathbf{x}_v, t_s, t_e) \in V_T\}$ and $\bar{t}_e(v) = \max\{t_e : (v, \mathbf{x}_v, t_s, t_e) \in V_T\}$ the time of first and last appearance of a node, and similarly for $\bar{t}_s(e)$, $\bar{t}_e(e)$, $e \in E$. Moreover, we set $T_s(G_T) := \min\{\bar{t}_s(v) : v \in V\}$, $T_e(G_T) := \max\{\bar{t}_e(v) : v \in V\}$ as the initial and final timestamps in a TG G_T. For two TGs, $G_T^i := (V^i, E^i, V_T^i, E_T^i)$, $i = 1, 2$, we write $G_T^1 \subseteq_V G_T^2$ to indicate the topological inclusion $V^1 \subseteq V^2$, while no relation between the corresponding timestamps is required.

Given $v \in V$, the set

$$\mathcal{N}_t[v] := \{u \in V : \exists\ (e, \mathbf{x}_e, t_s, t_e) \in E_T \text{ with } e = (u, v), t_s \leq t\}$$

is the temporal neighborhood of v at time t. i.e., the set of nodes that have been connected to v until time t.

General TGs have no restriction on their timestamps, which can take any value (for simplicity, we just assume that they are non-negative). However, in some applications, it makes sense to force these values to be multiples of a fixed time-step. This leads to the notion of Discrete Time Temporal Graphs, which are defined as follows.

Definition 3.1.2 (Discrete Time Temporal Graph - DTTG)**.** *Let $\Delta t > 0$ be a fixed time-step and let $t_1 < t_2 < \cdots < t_n$ be timestamps with $t_{k+1} = t_k + \Delta t$. A* Discrete Time Temporal Graph *G_{DT} is a TG where for each $(v, \mathbf{x}_v, t_s, t_e) \in V_T$ or $(e, \mathbf{x}_e, t_s, t_e) \in E_T$, the timestamps t_s, t_e are taken from the set of fixed timestamps (i.e., $t_s, t_e \in \{t_1, t_2, \ldots, t_n\}$, with $t_s < t_e$).*

3.1.1 Representation of temporal graphs

In the existing literature, dynamic graphs are often divided into DTTG (as in Definition 3.1.2) and continuous-time temporal graphs (CTTG) (or time sequence graphs), which are defined, e.g., in [83, 84, 85, 86]. However, we find that this separation does not capture well the central difference between various graph characterizations, which is rather based on the fact that the data are represented as a stream of static graphs, or as a stream of single node and edge addition and deletion events. We thus formalize the following two categories for the description of time-varying graphs, based on snapshots or on events. These different representations lead to different algorithmic approaches and become particularly useful when organizing the methods in a taxonomy.

The snapshot-based strategy focuses on the temporal evolution of the whole graph. Snapshot-based Temporal Graphs can be defined as follows.

Definition 3.1.3 (Snapshot-based Temporal Graph - STG)**.** *Let $t_1 < t_2 < \cdots < t_n$ be the ordered set of all timestamps t_s, t_e occurring in a TG G_T. Set*

$$V_i := \{(v, \mathbf{x}_v) : (v, \mathbf{x}_v, t_s, t_e) \in V_T, t_s \leq t_i \leq t_e\},$$
$$E_i := \{(e, \mathbf{x}_e) : (e, \mathbf{x}_e, t_s, t_e) \in E_T, t_s \leq t_i \leq t_e\},$$

and define the snapshots *$G_i := (V_i, E_i)$, $i = 1, \ldots, n$. Then a* Snapshot-based Temporal Graph *representation of G_T is the sequence*

$$G_T^S := \{G_i : i = 1, \ldots, n\}$$

of time-stamped static graphs.

This representation is mostly used to describe DTTGs, where the snapshots represent the TG captured at periodic intervals (e.g., hours, days, etc.).

The event-based strategy is instead more appropriate when the focus is on the temporal evolution of individual nodes or edges. This leads to the following definition.

Definition 3.1.4 (Event-based Temporal Graph - ETG)**.** *Let G_T be a TG, and let ε denote one of the following* events*:*

- Node insertion $\varepsilon_V^+ := (v, t)$*: the node v is added to G_T at time t, i.e., there exists $(v, \mathbf{x}_v, t_s, t_e) \in V_T$ with $t_s = t$.*
- Node deletion $\varepsilon_V^- := (v, t)$*: the node v is removed from G_T at time t, i.e., there exists $(v, \mathbf{x}_v, t_s, t_e) \in V_T$ with $t_e = t$.*
- Edge insertion $\varepsilon_E^+ := (e, t)$*: the edge e is added to G_T at time t, i.e., there exists $(e, \mathbf{x}_e, t_s, t_e) \in E_T$ with $t_s = t$.*
- Edge deletion $\varepsilon_E^- := (e, t)$*: the edge e is removed from G_T at time t, i.e., there exists $(e, \mathbf{x}_e, t_s, t_e) \in E_T$ with $t_e = t$.*

An Event-based Temporal Graph *representation of TG is a sequence of events*

$$G_T^E := \{\varepsilon : \varepsilon \in \{\varepsilon_V^+, \varepsilon_V^-, \varepsilon_E^+, \varepsilon_E^-\}\}.$$

Here it is implicitly assumed that node and edge events are consistent (e.g., a node deletion event implies the existence of an edge deletion event for each incident edge). In the case of an ETG, the TG structure can be recovered by coupling an insertion and deletion event for each temporal edge and node. ETGs are better suited than STGs to represent TGs with arbitrary timestamps.

We will use the general notion of TG, which comprises both STG and ETG, in formalizing learning tasks in the next section. On the other hand, we will revert to the STG and ETG notions when introducing the taxonomy of TGNN methods in Section 3.5, since TGNNs use one or the other representation strategy in their algorithmic approaches.

3.2 Learning Settings

Thanks to their learning capabilities, TGNNs are extremely flexible and can be adapted to a wide range of tasks on TGs. Some of these tasks are straightforward temporal extensions of their static counterparts. However, the temporal dimension has some non-trivial consequences in the definition of learning settings and tasks, some of which are often only loosely formalized in the literature. We start by formalizing the notions of transductive and inductive learning for TGNNs, and then describe the different tasks that can be addressed.

The machine learning literature distinguishes between inductive learning, in which a model is learned on training data and later applied to unseen test instances, and transductive learning, in which the input data of both training and test instances are assumed to be available, and learning is equivalent to leveraging the training inputs and labels to infer the labels of test instances given their inputs. This distinction becomes extremely relevant for graph-structured data, where the

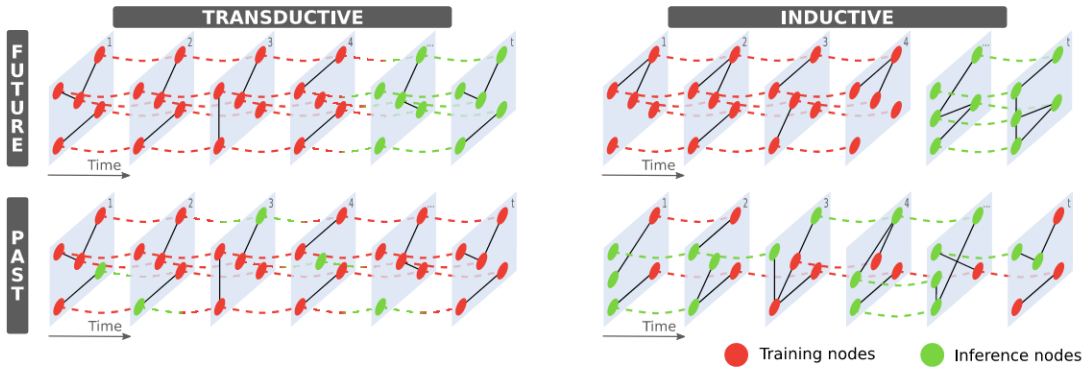


Figure 3.1: **Learning settings.** Schematic representation of the learning settings on TGs formalized in Section 3.2. The temporal graphs are represented as sequences of snapshots, with training (red) and inference (green) nodes connected by edges (solid lines), and where a dotted line connects instances of the same node (with possibly different features and/or labels) in successive snapshots. The four categories are obtained from the different combinations of a temporal and a topological dimension. The **temporal** dimension distinguishes the *future* setting, where the training nodes are all observed before the inference nodes (first row), from the *past* setting where inference is performed also on nodes appearing before the observation of the last training node (second row). The **topological** dimension comprises a *transductive* setting, where each inference node is observed (unlabelled) also during training (left column), and an *inductive* setting, where inference is performed on nodes that are unknown at training time (right column).

topological structure gives rise to a natural connection between nodes, and thus to a way to propagate the information in a transductive fashion. Roughly speaking, transductive learning is used in the graph learning literature when the node to be predicted and its neighborhood are known at training time — a typical situation in node classification tasks —, while inductive learning indicates that this information is not available — and is most often associated to graph classification tasks.

However, when talking about GNNs with their representation learning capabilities, this distinction is not so sharp. For example, a GNN trained for node classification in transductive mode could still be applied to an unseen graph, thus effectively performing inductive learning. The temporal dimension makes this classification even more elusive, since the graph structure is changing over time and nodes are naturally appearing and disappearing. Defining node membership in a temporal graph is thus a challenging task in itself.

Below, we provide a formal definition of transductive and inductive learning for TGNNs which is purely topological, i.e., linked to knowing or not the instance to be predicted at the training time, and we complete it with a temporal dimension, which distinguishes between past and future prediction tasks. A schematic representation of these settings is visualized in Figure 3.1. We recall (see Section 3.1) that $T_e(G_T)$ is the final timestamp in a TG G_T.

Definition 3.2.1 (Learning settings). *Assume that a model is trained on a set of $p \geq 1$ temporal*

graphs $\mathcal{G}_{\mathcal{T}} := \{G_T^i := (V_i, E_i, V_{T_i}, E_{T_i}),\ i = 1, \dots, p\}$. *Moreover, let*

$$T_e^{all} := \max_{i=1,\dots,p} T_e(G_T^i),\ V^{all} := \cup_{i=1}^{p} V_i,\ E^{all} := \cup_{i=1}^{p} E_i,$$

be the final timestamp and the set of all nodes and edges in the training set. Then, we have the following settings:

- Transductive learning*: inference can only be performed on* $v \in V^{all}$, $e \in E^{all}$, *or* $G_T \subseteq_V G_T^i$ *with* $G_T^i \in \mathcal{G}_{\mathcal{T}}$.
- Inductive learning*: inference can be performed also on* $v \notin V^{all}$, $e \notin E^{all}$, *or* $G_T \not\subseteq_V G_T^i$, *for all* $i = 1, \dots, p$.
- Past prediction*: inference is performed for* $t \leq T_e^{all}$.
- Future prediction*: inference is performed for* $t > T_e^{all}$.

We remark that all combinations of topological and temporal settings are meaningful, except for the case of inductive graph-based tasks. Indeed, the measure of time used in TGs is relative to each single graph. Moving to an unobserved graph would thus make the distinction between past and future pointless. Moreover, let us observe that, in all other cases, the two temporal settings are defined based on the final time of the entire training set, and not of the specific instances (nodes or edges), since their embedding may change also as an effect of the change of their neighbors in the training set.

We will use this categorization to describe supervised and unsupervised learning tasks in Sections 3.3-3.4, and to present existing models in Section 3.5.

3.3 Supervised Learning Tasks

Supervised learning tasks are based on a dataset where each object is annotated with its label (or class), from a finite set of possible choices $\mathcal{C} := \{C_1, C_2, \dots, C_k\}$.

3.3.1 Classification

Definition 3.3.1 (Temporal Node Classification)**.** *Given a TG* $G_T = (V, E, V_T, E_T)$, *the node classification task consists in learning the function*

$$f_{NC} : V \times \mathbb{R}^+ \to \mathcal{C}$$

which maps each node to a class $C \in \mathcal{C}$, *at a time* $t \in \mathbb{R}^+$.

This is one of the most common tasks in the TGNN literature. For instance, [87, 55, 59, 88, 58] focus on a future-transductive (FT) setting, i.e., predicting the label of a node in future timestamps. TGAT [55] performs future-inductive (FI) learning, i.e., it predicts the label of an unseen node in the future. Finally, DGNN [89] is the only method that has been tested on a past-inductive (PI) setting, i.e., predicting labels of past nodes that are unavailable (or masked) during training, while no approach has been applied to the past-transductive (PT) one. A significant application

may be in epidemic surveillance, where contact tracing is used to produce a TG of past human interactions, and sample testing reveals the labels (infection status) of a set of individuals. Identifying the past infection status of the untested nodes is a PT task.

Definition 3.3.2 (Temporal Edge Classification)**.** *Given a TG $G_T = (V, E, V_T, E_T)$, the temporal edge classification task consists in learning a function*

$$f_{EC} : E \times \mathbb{R}^+ \to \mathcal{C}$$

which assigns each edge to a class at a given time $t \in \mathbb{R}^+$.

Temporal edge classification has been less explored in the literature. Existing methods have focused on FT learning [87, 59], while FI, PI and PT have not been tackled so far. An example of PT learning consists in predicting the unknown past relationship between two acquaintances in a social network given their subsequent behavior. For FI, one may predict if a future transaction between new users is a fraud or not.

In the next definition we use the set of real and positive intervals $I^+ := \{[t_s, t_e] \subset \mathbb{R}^+\}$.

Definition 3.3.3 (Temporal Graph Classification)**.** *Let $\mathcal{G}_\mathcal{T}$ be a domain of TGs. The graph classification task requires to learn a function*

$$f_{GC} : \mathcal{G}_\mathcal{T} \times I^+ \to \mathcal{C}$$

that maps a temporal graph, restricted to a time interval $[t_s, t_e] \in I^+$, into a class.

The definition includes the classification of a single snapshot (i.e., $t_s = t_e$). As mentioned above, in the inductive setting the distinction between past and future predictions is pointless. In the transductive setting, instead, a graph $G_T \in \mathcal{G}_\mathcal{T}$ may be classified in a past mode if $[T_s(G_T), T_e(G_T)] \subseteq [t_s, t_e]$, or in the future mode, otherwise.

The only existing method addressing the classification of temporal graphs is found in [90], where the discrimination between STGs characterized by different dissemination processes is formalized as a PT classification task.

The temporal graph classification task can have numerous relevant applications. For instance, an example of inductive temporal graph classification is predicting mental disorders from the analysis of the brain connectome [91]. On the other hand, detecting critical stages during disease progression from gene expression profiles [92] can be framed as a past transductive graph classification task.

3.3.2 Regression

The tasks introduced for classification can all be turned into corresponding regression tasks, simply by replacing the categorical target $\mathcal{C}$ with the set $\mathbb{R}$. We omit the formal definitions for the sake of brevity. Static GNNs have already shown outstanding results in this setting, e.g., in weather forecasting [93] and earthquake location and estimation [94]. However, limited research has been conducted on the application of TGNNs to regression tasks. Notable exceptions are the use of TGNNs in two FT regression tasks, the traffic prediction [95] and the prediction of the incidence of chicken pox cases in neighboring countries [90].

3.3.3 Link Prediction

Link prediction requires the model to predict the relation between two given nodes, and can be formulated by taking as input any possible pair of nodes. Thus, we consider the setting to be transductive when both node instances are known at training time, and inductive otherwise. Instead, [96] adopts a different approach and identifies *Level-1* (the set of nodes is fixed) and *Level-2* (nodes may be added and removed over time) temporal link prediction tasks.

Definition 3.3.4 (Temporal Link Prediction)**.** *Let $G_T = (V, E, V_T, E_T)$ be a TG. The temporal link prediction task consists in learning a function*

$$f_{LP} : V \times V \times \mathbb{R}^+ \to [0, 1]$$

which predicts the probability that, at a certain time, there exists an edge between two given nodes.

The domain of the function f_{LP} is the set of all feasible pairs of nodes, since it is possible to predict the probability of future interactions between nodes that have been connected in the past or not, as well as the probability of missing edges in a past time. Most TGNN approaches for temporal link prediction focus on future predictions, forecasting the existence of an edge in a future timestamp between existing nodes (FT is the most common setting) [87, 57, 56, 97, 55, 85, 59, 89, 58, 88], or unseen nodes (FI) [56, 55, 58]. The only model that investigates past temporal link prediction is [85], which devises a PI setting by masking some nodes and predicting the existence of a past edge between them. Note that predicting past temporal links can be extremely useful for predicting, for instance, missing interactions in contact tracing for epidemiological studies.

Definition 3.3.5 (Event Time Prediction)**.** *Let $G_T = (V, E, V_T, E_T)$ be a TG. The aim of the event time prediction task consists in learning a function*

$$f_{EP} : V \times V \to \mathbb{R}^+$$

that predicts the time of the first appearance of an edge.

None of the existing methods address this task. Potential FT applications of event time prediction include predicting when a customer will pay an invoice to its supplier, or how long it takes to connect two similar users in a social network.

3.4 Unsupervised Learning Tasks

In this section, we formalize unsupervised learning tasks on temporal graphs, an area that has received little to no attention in the TGNN literature so far.

3.4.1 Clustering

Temporal graphs can be clustered at the node or graph level, with edge-level clustering being a minor variation of the node-level one. Some relevant applications can be defined in terms of temporal clustering.

Definition 3.4.1 (Temporal Node Clustering)**.** *Given a TG $G_T = (V, E, V_T, E_T)$, the temporal node clustering task consists in learning a time-dependent cluster assignment map*

$$f_{NCl} : V \times \mathbb{R}^+ \to \mathcal{P}(V),$$

where $\mathcal{P}(V) := \{p_1, p_2, \ldots, p_k\}$ is a partition of the node set V, i.e., $p_i \subset V_T$, $p_i \cap p_j = \emptyset$, if $i \neq j$, $\cup_{i=1}^{N} p_i = V_T$.

While node clustering in standard graphs is a very common task, its temporal counterpart has not been explored yet for TGNNs, despite its potential relevance in application domains like epidemic modelling — e.g., identifying groups of exposed individuals, in both inductive and transductive settings [98, 99, 100, 101, 102] —, trend detection in customer profiling — mostly in transductive settings [103, 104] —, or disease clustering — mostly in future transductive settings [105, 106, 107, 108].

Definition 3.4.2 (Temporal Graph Clustering)**.** *Given a set of temporal graphs $\mathcal{G}_\mathcal{T}$, the temporal graph clustering task consists in learning a cluster-assignment function*

$$f_{GCl} : \mathcal{G}_\mathcal{T} \times I^+ \to \mathcal{P}(\mathcal{G}_\mathcal{T}),$$

where $\mathcal{P}(\mathcal{G}_\mathcal{T}) := \{p_1, \ldots, p_k\}$ is a partition of the set of temporal graphs in the given time interval.

Relevant examples of tasks of inductive temporal graph clustering are grouping social interaction networks (e.g., hospitals, workplaces, schools) according to their interaction patterns [39, 46], or grouping diseases in terms of similarity between their spreading processes [109, 110].

3.4.2 Anomaly detection

Anomaly detection in TGs refers to the process of identifying any significant deviations from the expected or normal patterns of connectivity, behavior, or structural properties within the TG. These anomalies may indicate critical events, emerging trends, unusual behaviors, or potential threats. The basic idea is to model the behavior of the data using density estimation methods and then identify instances, which deviate significantly from this behavior, as anomalies. In the following, we identify and formalize three types of anomalies.

Definition 3.4.3 (Anomalous node detection)**.** *Given a TG $G_T = (V, E, V_T, E_T)$, a TGNN and an application dependent threshold $C \in (0, 1)$, a node $v \in V$ is detected as anomalous if*

$$f_{DE}(\texttt{TGNN}(v, G_T; \theta)) < C,$$

where f_{DE} is a procedure for density estimation like a variational autoencoder [111], a generative adversarial network [112] or a kernel density estimation function [113].

For example, detecting anomalous nodes within a TG can be applied to identify the origin nodes of a virus in a network. By comparing the activity of each node on the days surrounding a known virus attack to their activity on the day of the attack, the nodes that exhibit higher-than-usual activity at the time of the attack can be detected as the potential source of the virus [114]. Other applications include discovering anomalies in communication networks [115] and observing the shifts in community involvement [116, 117, 118].

Definition 3.4.4 (Anomalous edge detection)**.** *Given a TG $G_T = (V, E, V_T, E_T)$, a TGNN, an application dependent threshold $C \in (0, 1)$ and a procedure for density estimation f_{DE}, an edge $e \in E$ is detected as anomalous if*

$$f_{DE}(\texttt{TGNN}(e, G_T; \theta)) < C.$$

Edge anomaly detection finds extensive applications in various fields, such as the analysis of vehicle traffic patterns [119, 120] and the identification of improbable social interactions [121, 122].

Definition 3.4.5 (Anomalous graph detection)**.** *Given a domain of TGs $\mathcal{G}_\mathcal{T}$, a TGNN, an application dependent threshold $C \in (0, 1)$ and a procedure for density estimation f_{DE}, a TG $G_T \in \mathcal{G}_\mathcal{T}$ is detected as anomalous if*

$$f_{DE}(\texttt{TGNN}(G_T; \theta)) < C.$$

Possible application of anomalous graph detection are identifying accidents in TGs representing vehicle traffic [123, 124] as well as detecting whether a molecule is mutagenic [125].

3.4.3 Low-dimensional embedding (LDE)

LDEs are especially useful in the temporal setting, e.g., to visually inspect temporal dynamics of individual nodes or entire graphs, and identify relevant trends and patterns. No GNN-based models have been applied to these tasks, neither at the node nor at the graph level. We formally define the tasks of temporal node and graph LDE as follows.

Definition 3.4.6 (Low-dimensional temporal node embedding)**.** *Given a TG $G_T = (V, E, V_T, E_T)$, the low-dimensional temporal node embedding task consists in learning a map*

$$f_{NEm} : V \times \mathbb{R}^+ \to \mathbb{R}^d$$

to map a node, at a given time, into a low dimensional space.

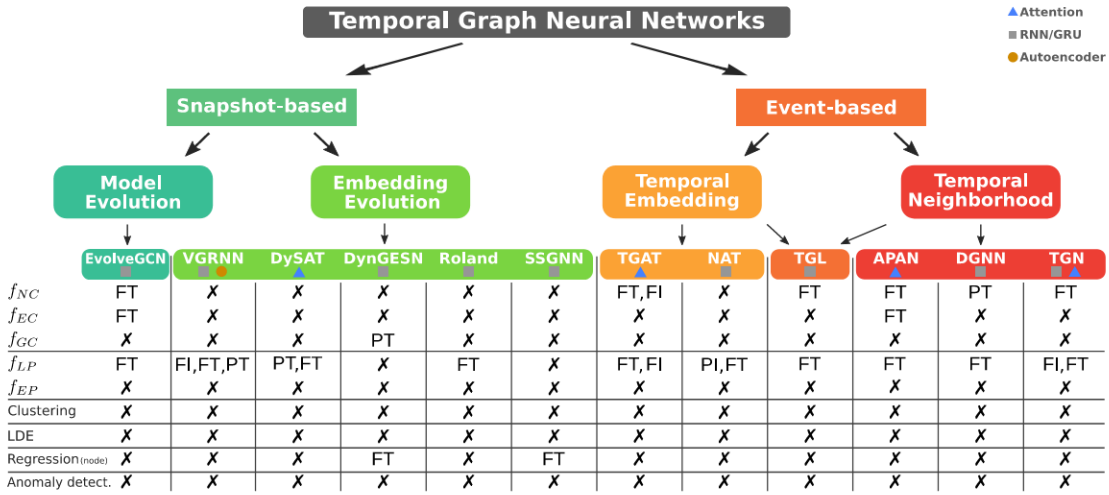
Definition 3.4.7 (Low-dimensional temporal graph embedding)**.** *Given a domain of TGs $\mathcal{G}_\mathcal{T}$, the low-dimensional temporal graph embedding task aims to learn a map*

$$f_{GEm} : \mathcal{G}_\mathcal{T} \times I^+ \to \mathbb{R}^d,$$

which represents each graph as a low dimensional vector in a given time interval.

3.5 Taxonomy of Temporal Graph Neural Networks

This section describes the taxonomy with which we categorize existing TGNN approaches (see Figure 3.2). All these methods learn a time-dependent embedding $\mathbf{h}_v(t) = \texttt{TGNN}(v, G_T; \theta)$ of each node $v \in V_T$ of a TG G_T, where θ represents a set of trainable parameters. Following the representation strategies outlined in Section 3.1, the first level groups methods into *Snapshot-based* and *Event-based*. The second level of the taxonomy further divides these two macro-categories



	EvolveGCN	VGRNN	DySAT	DynGESN	Roland	SSGNN	TGAT	NAT	TGL	APAN	DGNN	TGN
f_{NC}	FT	✗	✗	✗	✗	✗	FT,FI	✗	FT	FT	PT	FT
f_{EC}	FT	✗	✗	✗	✗	✗	✗	✗	✗	FT	✗	✗
f_{GC}	✗	✗	✗	PT	✗	✗	✗	✗	✗	✗	✗	✗
f_{LP}	FT	FI,FT,PT	PT,FT	✗	FT	✗	FT, FI	PI,FT	FT	FT	FT	FI,FT
f_{EP}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Clustering	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
LDE	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Regression(node)	✗	✗	✗	FT	✗	FT	✗	✗	✗	✗	✗	✗
Anomaly detect.	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

Figure 3.2: **The proposed TGNN taxonomy with an analysis of the surveyed methods.** The top panel shows the categories with the corresponding model instances (Section 3.5), where the colored bullets additionally indicate the main technology that they employ. The bottom table maps these methods to the task (Sections 3.3,3.4) to which they have been applied in the respective original paper, with an additional indication of their use in the future (F), past (P), inductive (I), or transductive (T) settings (Section 3.2). Note that no method has been applied yet to clustering and visualization, for neither graphs nor nodes. Moreover, only four out of ten models have been tested in the past mode (three in PT, one in PI).

based on the techniques used to manage the temporal dependencies. The leaves of the taxonomy in Figure 3.2 correspond to the individual models, with a colored symbol indicating their main underlying technology.

In the following we will denote as $\texttt{REC}(v_1, \ldots, v_k)$ a network that can process streams of tensors $v_1, \ldots, v_k$ and predict the next one in the sequence. This is usually a Recurrent Neural Network (from which we set the name `REC`), but other mechanisms, such as temporal attention, can be used.

3.5.1 Snapshot-based models

Snapshot-based models are specifically tailored for STGs (see Def. 3.1.3) and thus, consistently with the definition, they are equipped with a suitable method to process the entire graph at each point in time, and with a mechanism that learns the temporal dependencies across time-steps. Based on the mechanism used, we can further distinguish between *Model Evolution* and *Embedding Evolution* methods.

Model Evolution methods We call *Model Evolution* the evolution of the parameters of a static GNN model over time. This mechanism is appropriate for modelling STGs, as the evolution of the model is performed at the snapshot level.

More formally, these methods learn an embedding

$$\begin{aligned} \mathbf{x}_v^L(t_i) &= \texttt{GNN}(v, G_i; \theta(t_i)), \\ \theta(t_i) &= \texttt{REC}(\theta(t_{i-j}) : 1 \leq j \leq i_{\max}) \end{aligned} \tag{3.1}$$

where $\mathbf{x}_v^L(t_i)$ is the representation of node v produced after L layers of a GNN at time t_i, $\theta(t_i)$ is a parameter-evolution network, $i_{\max}$ is the memory length and REC is a recursive function.

To the best of our knowledge, the only existing method belonging to this category is **EvolveGCN** [87]. This model utilizes a Recurrent Neural Network (RNN) to update the Graph Convolutional Network (GCN) [14] parameters at each time-step, allowing for model adaptation that is not constrained by the presence or absence of nodes. The method can effectively handle new nodes without prior historical information. A key advantage of this approach is that the GCN parameters are no longer trained directly, but rather they are computed from the trained RNN, resulting in a more manageable model size that does not increase with the number of time-steps. The paper presents two versions of this method: EvolveGCN-O uses a Long Short-Term Memory (LSTM) to simply evolve the weights in time, while EvolveGCN-H represents the weights as hidden states of a Gated Recurrent Unit (GRU), whose input is the previous node embedding.

Embedding Evolution methods Rather than evolving the parameters of a static GNN model, *Embedding Evolution* methods focus on evolving the embeddings produced by a static model. Methods that belong to this category can be expressed as:

$$\begin{aligned} \mathbf{x}_v^L(t_i) &= \texttt{GNN}(v, G_i; \theta) \quad \forall i \geq 0 \\ \mathbf{q}_v(t_i) &= \texttt{REC}(\mathbf{q}_v(t_{i-1}), \mathbf{x}_v^L(t_i)) \quad \forall i > 0 \\ \mathbf{q}_v(t_0) &:= \mathbf{x}_v^L(t_0). \end{aligned} \tag{3.2}$$

Due to their simplicity, they rank among the most frequently employed methods for STGs, yielding substantial practical and theoretical outcomes. In the next section, we will illustrate an approximation theorem for this class of models, henceforth referred to as STGNNs. There are several different STGNN models in literature. These networks differ from one another in the techniques used for processing both the structural information and the temporal dynamics of the STGs.

DySAT [57] introduces a generalization of Graph Attention Network (GAT) [47] for STGs. First, it uses a self-attention mechanism to generate static node embeddings at each timestamp. Then, it uses a second self-attention block to process past temporal embeddings for a node to generate its novel embedding. Decoupling graph evolution into two modular blocks allows for efficient computations of temporal node representations. The structural and temporal self-attention layers, combined and stacked, enable flexibility and scalability.

The **VGRNN** model [56] uses VGAE [126] on each snapshot, where the latent representations are conditioned on a state variable modelled by Semi-Implicit Variational Inference (SIVI) [127] to handle the variation of the graph over time. The learned latent representation

is then evolved through an LSTM conditioned on the previous time's latent representation, allowing the model to predict the future evolution of the graph.

ROLAND [97] is a general framework for extending state-of-the-art GNN techniques to STGs. The key insight is that node embeddings at different GNN layers can be viewed as hierarchical node states. To generalize a static GNN for dynamic settings, hierarchical node states are updated based on newly observed nodes and edges through a Gated Recurrent Unit (GRU) update module [128]. The paper presents two versions of the model: ROLAND-MLP, which uses a 2-layer MLP to update node embeddings, and ROLAND-*moving average*, which updates the node embeddings through the moving average among previous node embeddings. Finally, reservoir computing techniques have also been proposed.

DynGESN [90] presents a method where each node embedding is updated by a recurrent mechanism using its temporal neighborhood and previous embedding, with fixed and randomly initialized recurrent weights.

SSGNN [95] follows a similar approach but introduces trainable parameters in the decoder and combines randomized components in the encoder: initially, the encoder creates representations of the time series data observed at each node, by utilizing a reservoir that captures dynamics at various time scales; these representations are then further processed to incorporate spatial dynamics dictated by the graph structure.

3.5.2 Event-based models

Models belonging to the *Event-based* macro category are designed to process ETGs (see Def. 3.1.4). These models are able to process streams of events by incorporating techniques that update the representation of a node whenever an event involving that node occurs, and represent an extension of message passing to TGs, since they combine and aggregate node representations over temporal neighborhoods. The models that lie in this macro category can be further classified in *Temporal Embedding* and *Temporal Neighborhood* methods, based on the technology used to learn the time dependencies. In particular, the *Temporal Embedding* models use recurrent or self-attention mechanisms to model sequential information from streams of events, while also incorporating a time encoding. This allows for temporal signals to be modeled by the interaction between time embedding, node features and the topology of the graph. *Temporal Neighborhood* models, instead, use a module that stores functions of events involving a specific node at a given time. These values are then aggregated and used to update the node representation as time progresses.

Temporal Embedding methods *Temporal embedding* methods model TGs by combining time embedding, node features, and graph topology. These models use an explicit functional time encoding, i.e., a vector embedding $\mathbf{g}_t$ of time based on Random Fourier Features (RFF) [129], which is translation-invariant (i.e., it depends only on the elapsed and not the absolute time).

They extend the message passing architecture to temporal neighborhoods, where the time is encoded by $\mathbf{g}_t$, i.e., at the l-th layer,

$$\mathbf{x}_v^l(t_i) = \texttt{COMBINE}^{(l)}((\mathbf{x}_v^{l-1}(t_i), \mathbf{g}_0), \texttt{AGGREGATE}^{(l)}(\{(\mathbf{x}_u^{l-1}(t_j), \mathbf{g}_{t_i - t_j}),\ u \in \mathcal{N}_{t_i}[v]\})),$$

where t_j is the time of the connection event between u and v. Thus, $\mathbf{g}_{t_i - t_j}$ encodes the time elapsed between the current time t and the time of connection between u and v.

TGAT [55], for example, introduces a graph-temporal attention mechanism which works on the embeddings of the temporal neighbors of a node, where the positional encoding is replaced by a temporal encoding based on RFFs. In addition, [55] implements a version of TGAT with all temporal attention weights set to an equal value (Const-TGAT). On the other hand, **NAT** [85] collects the temporal neighbors of each node into dictionaries, and then it learns the node representation with a recurrent mechanism, using the historical neighborhood of the current node and a RFF based time embedding. Note that [85] proposes a dedicated data structure to support parallel access and update of the dictionary on GPUs.

Temporal Neighborhood methods The *Temporal Neighborhood* class includes all TGNN models that make use of a special *mailbox* module to update node embeddings based on events. When an event ε occurs, a function is evaluated on the details of the event to compute a *mail* or a *message* $\mathbf{m}_\varepsilon$. For example, when a new edge appears between two nodes, a message is produced, taking into account the time of occurrence of the event, the node features, and the features of the new edge. The node representation is then updated at each time by aggregating all the generated messages. In more details, these methods extend message passing by learning at the l-th layer an embedding

$$\mathbf{x}_v^l(t_i) = \texttt{COMBINE}^{(l)}(\mathbf{x}_v^{l-1}(t_i), \texttt{AGGREGATE}^{(l)}(\{\mathbf{m}_\varepsilon,\, \varepsilon = (u, t_j) \text{ with } u \in \mathcal{N}_{t_i}[v]\})),$$

where ε, with $u \in \mathcal{N}_{t_i}[v]$, is the addition or deletion of a temporal neighbor of v.

Several existing TGNN methods belong to this category. **APAN** [59] introduces the concept of asynchronous algorithm, which decouples graph query and model inference. An attention-based encoder maps the content of the mailbox to a latent representation of each node, which is decoded by an MLP adapted to the downstream task. After each node update following an event, mails containing the current node embedding are sent to the mailboxes of its neighbors using a propagator. **DGNN** [89] combines an *interact* module — which generates an encoding of each event based on the current embedding of the interacting nodes and their history of past interactions — and a *propagate* module — which transmits the updated encoding to each neighbors of the interacting nodes. The aggregation of the current node encoding with those of its temporal neighbors uses a modified LSTM, which permits to work on non-constant time-steps, and implements a discount factor to downweight the importance of remote interactions. **TGN** [58] provides a generic framework for representation learning in ETGs, and makes an effort to integrate the concepts put forward in earlier techniques. This inductive framework is made up of separate and interchangeable modules. Each node the model has seen so far is characterized by a memory vector, which is a compressed representation of all its past interactions.

Given a new event, a mailbox module computes a mail for every node involved. Mails will then be used to update the memory vector. To overcome the so-called staleness problem [83], an embedding module computes, at each timestamp, the node embeddings using their neighborhood and their memory states. Finally, **TGL** [88] is a general framework for

training TGNNs on graphs with billions of nodes and edges by using a distributed training approach. In TGL, a mailbox module is used to store a limited number of the most recent interactions, called mails. When a new event occurs, the node memory of the relevant nodes is updated using the cached messages in the mailbox. The mailbox is then updated after the node embeddings are calculated. This process is also used during inference to ensure consistency in the node memory, even though updating the memory is not required during this phase.

3.5.3 Category comparison

The categories of models identified in our taxonomy exhibit various strengths, weaknesses, or suitability for specific scenarios. First and foremost, the comparison between the macro categories of Snapshot-based and Event-based methods is straightforward, hinging on the choice of temporal graph representation, namely STGs or ETGs. Within each macro category, sub-categories exhibit their own set of advantages and disadvantages. For instance, in the Model Evolution category, learning the evolution of GNN parameters becomes complex when the GNN has a large number of parameters. On the other hand, the Embedding Evolution category has a limitation in that temporal learning exclusively relies on recurrent mechanisms, which may not fully guarantee the preservation of temporal correlations among substructures. Despite this drawback, the approach offers simplicity and intuitiveness, allowing for the exploration and evaluation of various recurrent mechanisms.

In Temporal Embedding methods, defining the time encoding function is not trivial because it should capture different aspects of the graph, such as the temporal periodicity of interactions, recurrent interactions over time, and more. One advantage of this category, though, is that ad-hoc time encoding functions can be defined, depending on the application domain.

Finally, for the Temporal Neighborhood category, constructing the mailbox can be complex. For example, dense graphs have large mailboxes, which necessitate managing scalability issues. A specific mechanism to decide which nodes to include in the mailbox needs to be carefully designed, and this mechanism may also be domain-dependent. Similarly to the Temporal Embedding category, a benefit of the Temporal Neighborhood models is their ability to define domain-specific mailbox mechanisms.

In summary, each category of models for temporal graph learning has its own set of advantages and disadvantages. Choosing a category depends on the representation of the input graph, the complexity of learning the temporal dynamics, the need for domain-specific encoding or mailbox mechanisms, and scalability considerations.

3.6 Open Challenges

Building on existing libraries of GNN methods, two major TGNN libraries have been developed, namely PyTorch Geometric Temporal (PyGT) [130], based on PyTorch Geometric[1], and DynaGraph [131], based on Deep Graph Library[2]. While these are substantial contributions to the development and practical application of TGNN models, several open challenges still need to be

[1] https://pytorch-geometric.readthedocs.io
[2] https://docs.dgl.ai/

faced to fully exploit the potential of this technology. We discuss the ones we believe are the most relevant in the following.

Evaluation The evaluation of GNN models has been greatly enhanced by the Open Graph Benchmark (OGB) [132], which provides a standardized evaluation protocol and a collection of graph datasets enabling a fair and consistent comparison between GNN models. An equally well-founded standardized benchmark for evaluating TGNNs does not currently exist, even if a promising first step in this direction is the recently published Temporal Graph Benchmark (TGB)[3]. As a result, each model has been tested on its own selection of datasets, making it challenging to compare and rank different TGNNs on a fair basis. For instance, [88] introduced two real-world datasets with 0.2 billion and 1.3 billion temporal edges which allow to evaluate the scalability of TGNNs to large scale real-world scenarios, but only the TGL model was tested [88]. The variety and the complexity of learning settings and tasks described in Sections 3.2,3.3, 3.4 makes a standardization of tasks, datasets and processing pipelines especially crucial to allow a fair assessment of the different approaches and foster innovation in the field.

Another crucial aspect of evaluating GNN models is explainability, which is the ability to interpret and understand their decision process. While explainability has been largely explored for standard GNNs [133, 134, 135, 136], only few works focused on explaining TGNNs [137, 138, 139].

Learnability Training standard GNNs over large and complex graph data is highly non-trivial, often resulting in problems such as over-smoothing and over-squashing. A theoretical explanation for this difficulty has been given using algebraic topology and Sheaf theory [140, 141, 142]. More intuitively, we yet do not know how to reproduce the breakthrough obtained in training very deep architectures over vector data when training deep GNNs. Such a difficulty is even more challenging with TGNNs, because the typical long-term dependency of TGs poses additional problems to those due to over-smoothing and over-squashing.

Modern static GNN models face the problems arising from the complexity of the data using techniques such as dropout, virtual nodes, neighbor sampling, but a general solution is far from being reached. The extension of the above mentioned techniques to TGNNs, and the corresponding theoretical studies, are open challenges and we are aware of only one work towards this goal [143]. Moreover, the goal of proposing general very deep TGNNs is even more challenging, due to the difficulty in designing the graph dynamics in a hierarchical fashion.

Real-world applications The analysis of the tasks in Sections 3.3,3.4 revealed several opportunities for the use of TGNNs far beyond their current scope of application. We would like to outline here some promising directions of application. A challenging and potentially disruptive direction for the application of TGNNs is the learning of dynamical systems through the combination of machine learning and physical knowledge [144]. Physic Informed Neural Networks (PINNs) [145] are already revolutionizing the field of scientific computing [146], and static GNNs have been employed in this framework with great success [147, 148]. Adapting TGNNs to this field may enable to carry over these results to the

[3] https://tgb.complexdatalab.com/

treatment of time-dependent problems. Climate science [149] is a particularly attractive field of application, both for its critical impact in our societies and for the promising results achieved by GNNs in climate modelling tasks [93]. We believe that TGNNs may rise to be a prominent technology in this field, thanks to their unique capability to capture spatio-temporal correlations at multiple scales. Epidemics studies are another topic of enormous everyday impact that may be explored through the lens of TGNNs, since a proper modelling of the spreading dynamics needs to be tightly coupled to the underlying TG structure [109]. Both fields require a better development of TGNNs for regression problems, a task that is still underdeveloped (see Section 3.3). Another promising application for TGNNs is in relational databases [150], particularly those that incorporate temporal information [151].

Expressiveness Driven by the popularity of (static) GNNs, the study of their expressive power has received a lot of attention in the last few years [152]. Conversely, the expressive power of TGNNs is still far from being fully explored [153]. In the following section, we will show some findings regarding the expressiveness of TGNNs, as proposed in [66].

3.7 The Expressive Power of Temporal Graph Neural Networks

The expressiveness of TGNNs has yet to be fully exploited, and the design of new WL tests, suitable for TGNNs, is a crucial step towards this aim. This is a challenging task since the definition of a node neighborhood in temporal graphs is not as trivial as for static graphs, due to the appearing/disappearing of nodes and edges.

In the following, we will introduce an extension of the WL test for STGs, namely the DWL, and present a universal approximation theorem for embedding evolution STGNNs as defined in 3.2.

3.7.1 Dynamic WL test

Definition 3.7.1 (DWL)**.** *Let $G_T^S = \{G_i : i = 1, \dots, n\}$ as defined in Def. 3.1.3, with V the set of all nodes that appear at least in one timestamp. The DWL is defined as follows:*

- *Let $\texttt{HASH}_{t_i}^0$ be an injective time dependent function; in the first iteration $l = 0$ and for each time t_i, the color of node $v \in V$ is set to the hashed node feature or, whether v does not exist in that timestamp, to a fixed color $c^\perp$:*

$$c_v^0(t_i) = \begin{cases} \texttt{HASH}_{t_i}^0(\mathbf{x}_v) & \text{if } (v, \mathbf{x}_v) \in V_i, \\ c^\perp & \text{otherwise.} \end{cases}$$

- *In each iteration $l \geq 0$, the node color is updated using an injective function $\texttt{HASH}_{t_i}$:*

$$c_v^l(t_i) = \texttt{HASH}_{t_i}\left(c_v^{l-1}(t_i), \{c_u^{l-1}(t_i)\}_{u \in \mathcal{N}_{t_i}[v]}\right).$$

At the end of the DWL, to each node $v \in V$ is associated a vector $\mathbf{c}_v$ of colors, one for each t_i with $i = 1, ..., n$, i.e., $\mathbf{c}_v := [c_v^{L_i}(t_i)]_{i=1,\dots,n}$, being L_i the iteration in which the test converges at time t_i.

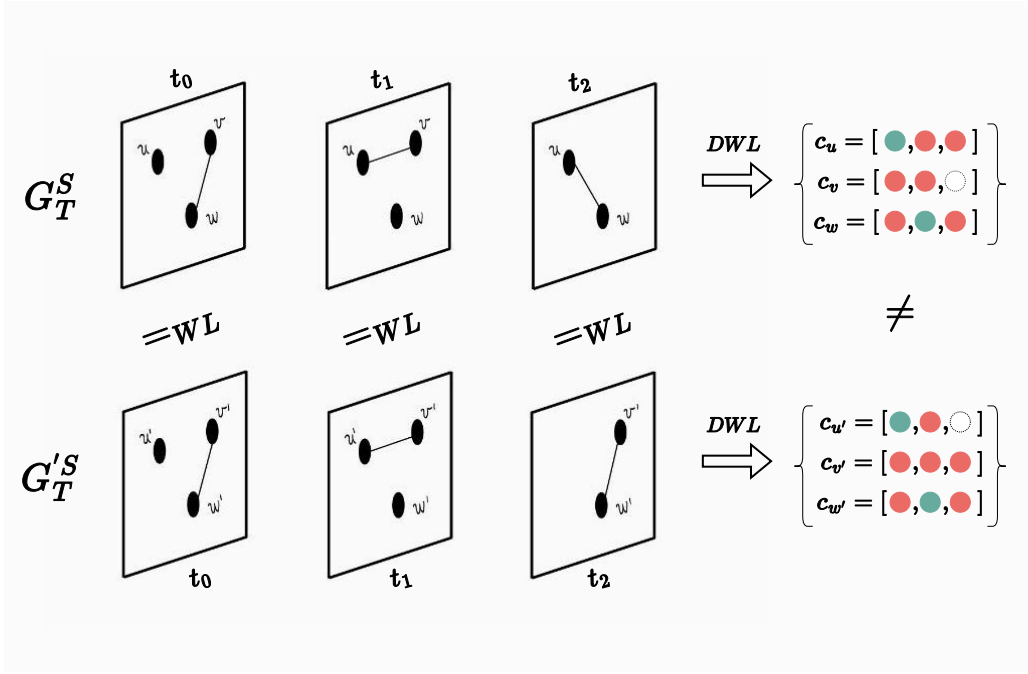


Figure 3.3: Two STGs that are WL equivalent in each snapshots but DWL not equivalent.

Note that, if a node does not exist at time t_i, its color is $c_v^l(t_i) = c^\perp$ for every $l \geq 0$ because its neighborhood is empty. This implies that non-existent nodes cannot influence the neighborhood aggregation of other nodes, or equivalently, that they cannot propagate through the graph.

Definition 3.7.2 (DWL equivalence)**.** *Two nodes $u, v \in V$ are said to be DWL equivalent, $u =_{DWL} v$, if $\mathbf{c}_v = \mathbf{c}_u$, that is, if their colors are equal in any timestamps. Analogously, let G_T^S and $G_T'^S$ be STGs with nodes sets V and W respectively. Then $G_T^S =_{DWL} G_T'^S$, if and only if $\{\mathbf{c}_u \mid u \in V\} = \{\mathbf{c}_v \mid v \in W\}$.*

Note that the DWL test is stronger than just stucking the outputs of the WL on the static graph snapshots. Indeed, the graph snapshots are not independent from each others, because nodes in different snapshots represent different instances of the same entity. When comparing each snapshots with the standard static WL test, the node entity does not affect the coloring process. When using the DWL, instead, for each node a vector of colors is produced, where each component represents the color of *that* particular node at a particular timestamp. As an example, consider the two STGs $G_T^S = \{G_1, G_2, G_3\}$ and $G_T'^S = \{G_1', G_2', G_3'\}$ in Figure 3.3. $G_i =_{WL} G_i'$ $\forall i$, whereas $G_T^S \neq_{DWL} G_T'^S$. Indeed, node $v' \in V'$ has one neighbour in every timestamp while no nodes in V exhibit the same pattern of number of neighbours, meaning the two multisets of node colors of the two graphs cannot be equal.

3.7.2 Universal Approximation Theorem for Temporal Graph Neural Networks

Definition 3.7.3 (Dynamic systems on STGs)**.** *Let $\mathcal{G}_\mathcal{T}$ be a domain of STGs and V^{all} be the set of all their nodes, as defined in 3.2.1. A dynamic system on STGs is a function $dyn : \{t_1, ..., t_n\} \times \mathcal{G}_\mathcal{T} \times V^{all} \to \mathbb{R}^m$ defined as:*

$$dyn(t_i, G_T^S, v) \;=\; \begin{cases} a(t_i, G_T^S, v) & \text{if } t_i = 0 \\ f(dyn(t_{i-1}, G_T^S, v), a(t_i, G_T^S, v)) & \text{if } t_i > 0 \end{cases}$$

where $a : \{t_1, ..., t_n\} \times \mathcal{G}_\mathcal{T} \times V^{all} \to \mathbb{R}^m$ is a function that processes the graph snapshot at time t_i and provides an m-dimensional internal state representation for each node v. Finally, $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ is the state update function.

Definition 3.7.4 (Preserving DWL equivalence)**.** *A dynamic system $dyn(\cdot, \cdot, \cdot)$ preserves DWL equivalence on $\mathcal{G}_\mathcal{T}$ if and only if, for any $G_T^S, G_T^{'S} \in \mathcal{G}_\mathcal{T}$ and for any two nodes $v \in V, u \in V^{'}$, it holds:*

$$v =_{DWL} u \implies dyn(t_i, G_T^S, v) = dyn(t_i, G_T^{'S}, u) \quad \forall i.$$

The class of dynamic systems that preserves the DWL equivalence on $\mathcal{G}_\mathcal{T}$ will be denoted with $\mathcal{F}(\mathcal{G}_\mathcal{T})$.

Theorem 1 (Universal Approximation Theorem)**.** *Given*

- $G_T^S \in \mathcal{G}_\mathcal{T}$*, with* $N = \max_{G_T^S \in \mathcal{G}_\mathcal{T}} |G_T^S|$*,*
- $dyn(t_i, G_T^S, v) \in \mathcal{F}(\mathcal{G}_\mathcal{T})$ *measurable dynamical system preserving the DWL equivalence,*
- P *probability measure on* $\mathcal{G}_\mathcal{T}$*,*
- $\epsilon, \lambda \geq 0$*,*

then, there exists a STGNN composed by a GNN with $2N - 1$ layers and hidden dimension $r = 1$, and with a state dimension $m = 1$, such that:

$$P(\|dyn(t_i, G_T^S, v) - STGNN(t_i, G_T^S, v)\| \leq \varepsilon) \geq 1 - \lambda, \quad \forall\, t_i.$$

The proof can be found in Appendix A
Theorem 1 intuitively states that, given a dynamical system that preserves the DWL equivalence, there is a STGNN that approximates it.
The functions which the STGNN is a composition of (such as the recursive function `REC`, $\texttt{COMBINE}^{(l)}$, $\texttt{AGGREGATE}^{(l)}$, etc.) are supposed to be continuously differentiable, but still generic and completely unconstrained. This situation does not correspond to practical cases where the STGNN adopts particular architectures, and those functions are parametric models — for example, made of layers of sum, max, average, etc. However, as proved in [154], if the STGNN incorporates neural networks whose universality has been established, specifically MLP [155] and RNN [156], then Thm. 1 holds.
The following remarks may further help to understand the results proven in this section.

- The proof of Thm. 1 is based on a rationale founded on space partitioning. Differently from the technique based on the Stone–Weierstrass theorem [157], which is existential in nature, such an approach allows us to deduce information about the characteristics of networks that reach the desired approximation. Actually, the theorem points out that the approximation can be obtained with minimal hidden and state dimensions, $r = 1$ and $m = 1$.

- Moreover, Thm. 1 specifies that GNNs can obtain the approximation with $2N - 1$ layers. We could incorrectly presume that the maximum number of layers required to reach a desired approximation depends on the diameter, $diam(G)$, of the graph, which can be smaller than the number of nodes N, since the information in a GNN can flow from one node to another in $diam(G)$ iterations. However, $diam(G)$ layers are not always sufficient to distinguish all the nodes of a graph. In fact, it has been proven that $N - 1$ is a lower bound on the number of iterations that the WL algorithm has to carry out to be able to distinguish any pairs of WL distinguishable graphs [158], and $2N - 1$ is a lower bound for WL algorithm to distinguish pairs of nodes in two different graphs [159]. Therefore, $2N - 1$ is also the lower bound for the GNN computation time to approximate any function for either graph-focused or node-focused tasks (see [154] for a detailed discussion).

3.7.3 Experimental Results

In order to validate our theoretical findings, we carried out two sets of experiments, described below.

E1. We show that an STGNN with universal components can approximate a function $F_{DWL} : \mathcal{G}_{\mathcal{T}} \to \mathbb{N}$ that models the DWL test. The function F_{DWL} assigns a target label to the input graph that represents the class of DWL equivalence.

E2. In the same approximation task, we compare STGNNs with different GNN modules from the literature to show how the universality of the components affects the approximation capability.

We focus on the ability of the STGNN to approximate F_{DWL}, so that only the training performance is considered, i.e., we do not investigate the generalization capabilities over a test set.

Dataset The dataset chosen consists of STGs with maximum timestamp $n = T$. Each static snapshot is one of the graphs in Fig. 3.4. Since the dataset is composed of all the possible combinations of the four graphs, it contains 4^T STGs. The graphs in Fig. 3.4 are pairwise WL equivalent, i.e., a) is WL equivalent to b) and c) is WL equivalent to d). Thus, the number of classes is 2^T, with $\frac{4^T}{2^T} = 2^T$ graphs in each class. For each STG, the target is the corresponding DWL output, represented as a natural number. For training purposes, the targets are normalized between 0 and 1 and uniformly spaced in the interval $[0, 1]$. Therefore, the distance between each class label is $d = 1/2^T$. An STG G_T^S with target $y_{G_T^S}$ will be said to be correctly classified if, given $\mathsf{out} = \text{STGNN}(G_T^S)$, we have $|\mathsf{out} - y_{G_T^S}| < d/2$.

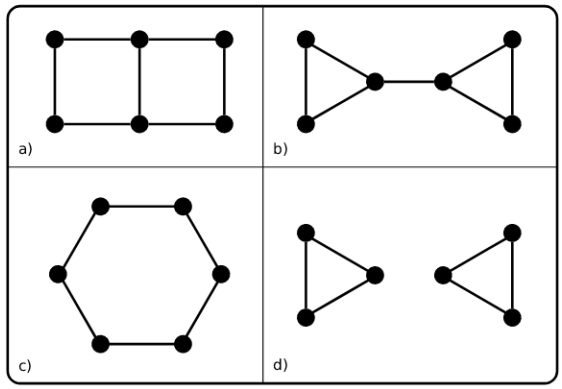


Figure 3.4: The four static graphs are used as components to generate the synthetic dataset. Graphs a) and b) are equivalent under the static WL test; the same holds for c) and d).

Experimental setup

E1. For the first set of experiments, the STGNN is composed of a GIN and an RNN, which implement the static GNN and the recurrent function `REC` of Eq. (3.2), respectively. We show that such an STGNN can approximate any dynamical system on the STG domain that preserve the DWL equivalence. The model hyperparameters for the experiments are set as follows. The GIN includes $n_{\max} = 6$ layers. The MLP in the GIN network contains one hidden layer with a hyperbolic tangent activation function and batch normalization. Hidden layers of different sizes, i.e., $h_{\mathsf{gin}} \in \{1, 4, 8\}$, have been tested. For the sake of simplicity, the output network has one hidden layer with the same number of neurons as the MLP in the GIN. Furthermore, $h_{\mathsf{rnn}} = 8$ is the size of the hidden state of the RNN.

E2. In the second set of experiments, we test STGNNs composed by different GNN static modules and an RNN module (analogously to **E1.**). In particular, we compare STGNNs with the GNN module taken from the following list:

- GIN;
- Graph Convolutional Network (GCN) [14];
- GNN presented in [160] (see also [74]), where the aggregation function is the *sum* of the hidden features of the neighbours; we will call this model `gconv_add`;
- GNN presented in [160] with *mean* of the hidden features of the neighbours as aggregation function; we will call this model `gconv_mean`;
- Graph Attention Network (GAT) [161].

The hidden dimension is $h = 8$, the number of layers is $L = 4$, and the time length is $T = 5$.

In both the experimental cases, the model is trained over 300 epochs using the Adam optimizer with an initial learning rate $\lambda = 10^{-3}$. Each configuration is is evaluated over 10 runs. The overall training is then performed on an Intel(R) Core(TM) i7-9800X processor, running at 3.80GHz using 31GB of RAM, and a GeForce GTX 1080 Ti GPU unit. The code used to run the experiments can be found at `https://github.com/AleDinve/dyn-gnn`.

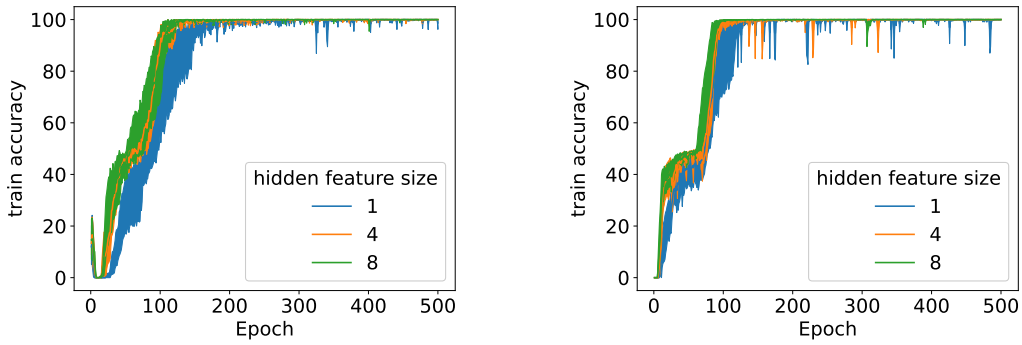


Figure 3.5: Experimental Framework **E1.** Training accuracy over the epochs for an STGNN trained on the dataset containing STGs up to time length $T = 4$ (a) and $T = 5$ (b).

Results The results of the experiments confirm our theoretical statements. More precisely, the STGNNs performed as follows during training.

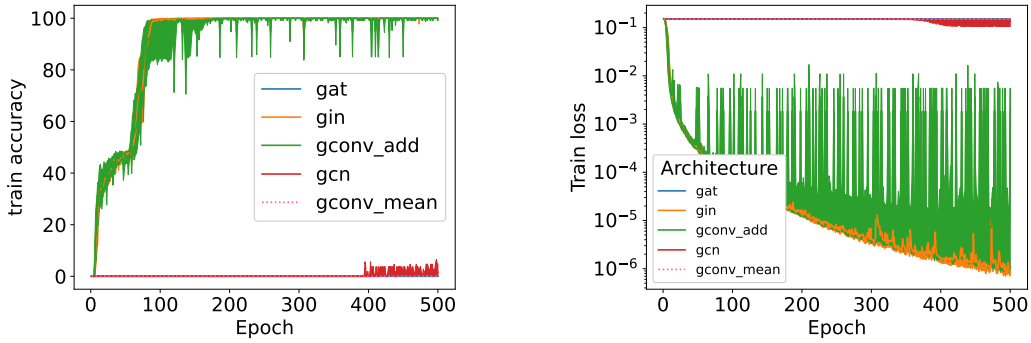


Figure 3.6: Experimental framework **E2.** Training accuracy a) and training loss b) over the epochs for several STGNNs trained on the dataset containing STGs up to time length $T = 5$. Figure b) is in logarithmic scale.

E1. In Fig. 3.5, the evolution of the training accuracy over the epochs is presented for different GIN hidden layer sizes h_{gin} and for STGs up to time lengths $T = 4$ (Fig. 3.5a) and $T = 5$ (Fig. 3.5b). All the architectures reach 100% accuracy for the experiments on both time lengths. Even setting $h_{\text{gin}} = 1$ leads to a perfect classification at a slower rate. It may appear surprising that, even with a hidden representation of size 1, the STGNN can precisely approximate the function F_{DWL}.

E2. The STGNN with the GIN module achieves the best performance in terms of learning accuracy and speed of decreasing of the loss function, as illustrated in Fig. 3.6. The STGNN with the gconv_add module is able to learn the task, although learning is unstable

(see Fig. 3.6 b)). This is not surprising since this module has been proven to match the expressive power of the WL test [74]. The other STGNNs are incapable to learn the objective function. This is a consequence of their weaker expressive power, widely investigated in literature [28, 154].

3.8 Other approaches to model temporal graphs

Many different representation techniques for temporal graphs, different from GNNs, have been proposed.

A popular class of approaches for learning an embedding function for temporal graphs is constituted by random walk-based methods. For example, in [162], temporal random walks are exploited to efficiently and automatically sample temporal network motifs, i.e., connected subgraphs with links that appear within a restricted time range. Similarly, in [163], a time-reinforced random walk is proposed to effectively sample the structural and temporal contexts over graph evolution. Also, [164] employs spatiotemporal-biased random walks to identify a collection of representative motifs, enabling the effective characterization of temporal nodes. With DyANE [165], temporal graphs are transformed into a static graph representation, called a supra-adjacency representation. In this approach, nodes are defined as (node, time) pairs from the original temporal graph. This static graph representation retains the temporal paths of the original network, which are crucial for comprehending and constraining the underlying dynamical processes. Afterwards, standard embedding techniques for static graphs, utilizing random walks, are employed.

Temporal graph learning has leveraged the use of temporal point processes as well. Temporal point processes are stochastic processes employed for modeling sequential asynchronous discrete events occurring in continuous time [166]. DyRep [167] is capable of learning a set of functions that can effectively generate evolving, low-dimensional node embeddings. By using the obtained node embeddings, a temporal point process is employed to estimate the likelihood of an edge connecting two nodes at a certain timestamp. In [168], instead, the occurrence of an edge in a temporal graph is modeled as a multivariate point process, where the intensity function is influenced by the score assigned to that edge, which is computed using the learned entity embeddings. The entity embeddings, which evolve over time, are acquired through a recurrent architecture.

Non-negative matrix factorization (NMF) has been employed for link prediction in temporal graphs. In [45], novel iterative rules for NMF are proposed to construct the matrix factors that capture crucial features of the temporal graph, enhancing the accuracy of the link prediction process.

Moreover, statistical approaches have been applied to TGs; for example, [169] introduces a novel whiteness hypothesis test specifically designed for spatio-temporal graphs. The test extends traditional methods used for system identification within graph signals to detect dependencies among temporal observations and spatial dependencies among graph neighborhoods. The test can also be used to assess the optimality of forecasting models.

Lastly, the majority of recently proposed methods employ deep learning techniques. For example, DynGem [170] is a dynamical autoencoder for growing graphs that construct the embedding of a snapshot based on the embedding of the previous snapshot. TRRN [171], instead,

uses multi-head self-attention to process a set of memories, enabling efficient information flow from past observations to current latent representations through shortcut paths. It incorporates policy networks with differentiable binary routers to estimate the activation probability of each memory and dynamically update them at the most relevant time-steps. In [172], a spatio-temporal attentive recurrent network model, called STAR, is proposed for interpretable temporal node classification. In [173], a node-level regression task is achieved by training embeddings to maximize the mutual information between patches of the graph, at any given time-step, and between features of the central nodes of patches, in the future. In [174], a spectral-based solution for learning representations of long-range interactions is proposed. Efficient spectral transforms and graph convolutions are employed to capture temporal features and interactions. The approach addresses challenges in ETG learning and achieves well-conditioned embeddings with minimal information loss. TSNet [175] is a comprehensive framework for node classification in temporal graphs, consisting of two key steps. Firstly, the graph snapshots undergo a sparsification process using edge sampling, guided by a learned distribution derived from the supervised classification task. This step effectively reduces the density of the snapshots. Subsequently, the sparsified snapshots are aggregated and processed through a convolutional network to extract meaningful features for node classification. In [176], a novel class of attention-based architectures, called Spatiotemporal Point Inference Network (SPIN), is introduced for addressing the challenge of reconstructing multivariate time-series on sparse graphs with missing data. SPIN exploits a spatiotemporal propagation process to learn predictive representations of unobserved samples, taking into account the data missingness. By incorporating a hierarchical attention mechanism, the proposed method reduces the space and time complexities involved.

3.9 Conclusions

In conclusion, GNNs have demonstrated their power and effectiveness as tools for processing different types of temporal graphs. However, they still remain partially unexplored in various settings and significant tasks where their full potential has yet to be tapped. This underexplored territory presents a rich field for future research, suggesting that much remains to be discovered about the capabilities and applications of GNNs in complex graph-based data environments.

Chapter 4

The Expressive Power of Pooling in Graph Neural Networks

Significant effort has been devoted to characterizing the expressive power of GNNs in terms of their capabilities for testing graph isomorphism [177]. This has led to a better understanding of the strengths and weaknesses of GNNs and opened up new avenues for developing advanced GNN models that go beyond the limitations of the current approaches [152]. The more powerful a GNN, the larger the set of non-isomorphic graphs that it can distinguish by generating distinct representations for them.

Despite the progress made in understanding the expressive power of GNNs, the results are still limited to *flat* GNNs, consisting of a stack of message-passing (MP) layers followed by a final readout [28, 178]. Inspired by pooling in convolutional neural networks, recent works introduced hierarchical pooling operators that enable GNNs to learn increasingly abstract and coarser representations of input graphs [64, 179]. By interleaving MP with pooling layers that gradually distill global graph properties through the computation of local summaries, it is possible to build deep GNNs that improve the accuracy in graph classification [63, 180] and node classification tasks [181, 182].

It is not straightforward to evaluate the power of a graph pooling operator and the quality of the coarsened graphs it produces. The most common approach is to simply measure the performance of a GNN with pooling layers on a downstream task, such as graph classification. However, such an approach is highly empirical and provides an indirect evaluation, affected by external factors. One issue consists in the overall GNN architecture: pooling is combined with different MP layers, activation functions, normalization or dropout layers, and optimization algorithms, which makes it difficult to disentangle the contribution of the individual components. Another factor is the dataset at hand: some classification tasks only require isolating a specific motif in the graph [183, 75], while others require considering global properties that depend on the whole graph structure [184]. Recently, two criteria were proposed to evaluate a graph pooling operator in terms of i) the spectral similarity between the original and the coarsened graph topology and ii) its capability of reconstructing the features of the original graph from the coarsened one [1]. While providing valuable insights, these criteria give results that are, to some extent, contrasting and in disagreement with the traditional evaluation based on the performance of the downstream task.

To address this issue, we introduce a universal and principled criterion that quantifies the power of a pooling operator as its capability to retain the information in the graph from an expressiveness perspective. In particular, we investigate how graph pooling affects the expressive power of GNNs and derive sufficient conditions under which the pooling operator preserves the highest degree of expressiveness.

This chapter will first establish the fundamental definitions and concepts of pooling. Following that, we show that, when certain conditions are met in the MP layers and in the pooling operator, their combination produces an injective function between graphs. This implies that the GNN can effectively coarsen the graph to learn high-level data descriptors, without compromising its expressive power. Moreover, based on our theoretical analysis, we identify commonly used pooling operators that do not satisfy these conditions and may lead to failures in certain scenarios. We introduce a simple yet effective experimental setup for measuring, empirically, the expressive power of *any* GNN in terms of its capability to perform a graph isomorphism test. Finally, we identify conditions under which expressiveness is not only maintained but also enhanced. We introduce a novel hierarchical pooling technique, named XPool, specifically designed to meet the necessary criteria for enhancing the expressive power of GNNs. Through rigorous testing on synthetic datasets aimed at evaluating the GNN expressiveness, our findings indicate that the XPool method significantly increases the expressive power of GNNs.

The results presented in this chapter are mainly based on the papers [65, 67].

4.1 Pooling in Graph Neural Networks: Basic Concepts and Definitions

A graph pooling operator implements a function $\texttt{POOL} : G \mapsto G_P = (V_P, E_P)$ such that $|V_P| = K$, with $K \leq N$. Let $\mathbf{X}_P \in \mathbb{R}^{K \times F}$ be the pooled node features, i.e., the features of the nodes V_P in the pooled graph (Figure 4.1). To formally describe the POOL function, we adopt the Select-Reduce-Connect (SRC) framework [1].

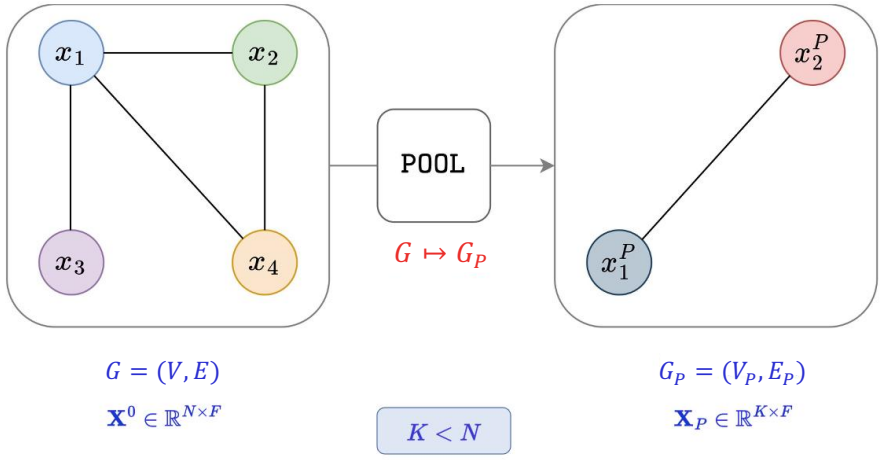


Figure 4.1: A schematic representation of a graph pooling operator.

4.1.1 Select, Reduce, Connect

A graph pooling operator ca be expressed through the combination of three functions: *selection*, *reduction*, and *connection* (Figure 4.2).

- The selection function (`SEL`) clusters the nodes of the input graph into subsets called *supernodes*, namely $\texttt{SEL} : G \mapsto \mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_K\}$ with $\mathcal{S}_j = \left\{ s_i^j \right\}_{i=1}^{N}$, where s_i^j is the membership score of node i to supernode j. The memberships are conveniently represented by a cluster assignment matrix $\mathbf{S}$, with entries $[\mathbf{S}]_{ij} = s_i^j$. Typically, a node can be assigned to zero, one, or several supernodes, each with different scores.

- The reduction function (`RED`) creates the pooled node features by aggregating the features of the nodes assigned to the same supernode, that is, $\texttt{RED} : (G, \mathbf{S}) \mapsto \mathbf{X}_P$.

- The connect function (`CON`) generates the edges and edge features, if applicable, by connecting the supernodes, i.e., $\texttt{CON} : (G, \mathcal{S}) \mapsto E_P$.

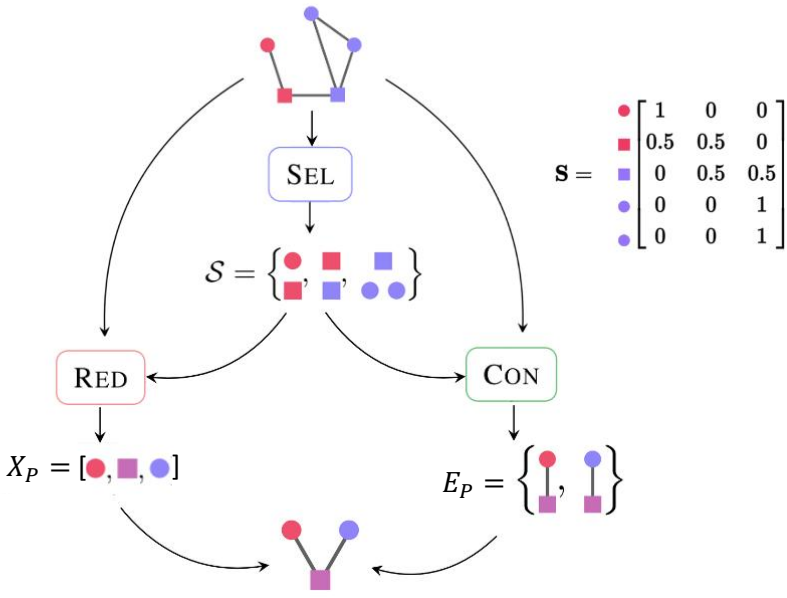


Figure 4.2: A schematic representation of the SRC framework. Adapted from [1].

4.1.2 Taxonomy of Graph Pooling

Based on the SRC framework, it is possible to categorize all the existing pooling operators. In particular, in [1], a taxonomy of pooling operators based on four distinguishing characteristics is proposed. A schematic representation of the branches of the taxonomy is proposed in Figure 4.3.

Trainability The first criterion in classifying pooling operators revolves around whether the `SEL`, `RED` and `CON` functions are integrated and learned together with the overall structure of the GNN. When these components are part of the end-to-end learning process in a GNN, the method is considered *trainable*. This implies that the operator's parameters are fine-tuned through a process that optimizes a task-specific loss function. In contrast, methods that do not follow this learning model are categorized as *not-trainable*. This distinction holds critical importance: not-trainable methods are commonly used as independent algorithms for the purpose of graph coarsening, while trainable methods were developed specifically for use in GNNs, marking them as an innovative area of research.

Typically, not-trainable methods are preferred when there is definitive prior knowledge regarding the aim of pooling, such as maintaining connectivity or selectively filtering certain graph frequencies. These methods are often built upon graph-theoretical properties and are especially useful in scenarios with limited data, as they do not contribute to increase the overall parameter count of the GNN and do not add extra objectives in the training process. A classic example of this situation is the standard grid pooling in CNNs. Trainable methods, in contrast, offer more adaptability and operate with fewer expectations about the outcome. This makes them particularly valuable in situations where the most effective pooling strategy is not known. However, it is also possible to incorporate predetermined assumptions about pooling in trainable methods. For instance, the MinCutPool [63] operator not only adapts to the GNN but also works towards a normalized cut objective to ensure that supernodes represent distinct clusters of nodes of comparable size. Such assumption serves as a regulatory mechanism in the optimization process of GNNs.

Density of Supernodes The taxonomy second dimension focuses on the size of supernodes and the associated computational cost for determining the `SEL` function. We quantify a pooling operator's density by the expected ratio $E(\frac{|\mathcal{S}_k|}{N})$, where $|\mathcal{S}_k|$ denotes the size of a supernode $\mathcal{S}_k$, and N represents the total node count in graph G.

A method is considered *dense* if the `SEL` function creates supernodes $\mathcal{S}_k$ with a size proportional to N, and it is deemed *sparse* when the supernode size remains constant at $O(1)$. This differentiation is crucial as sparse methods demand significantly fewer computational resources, notably in terms of memory, which remains a critical limitation in contemporary GPUs. This allows sparse methods to more effectively scale to larger graphs.

Adaptability Pooling methods can also be differentiated based on the number of nodes K of the pooled graph. When K is a constant value, independent of the size of the input graph, the pooling method is called *fixed*. In this scenario, K is a hyperparameter of the pooling operator, resulting in an output graph that consistently contains K nodes. For instance, K might be equivalent to the number of output features of a neural network that determines cluster assignments.

Alternatively, if the number of supernodes is determined by a function $K(G)$ of the input graph, the method is called *adaptive*. Often, $K(G)$ is related to the number of nodes N of the original graph, but it can be also dependent on the input graph in more complex ways. If maintaining relative graph size is critical for a specific task, adaptive methods are more suitable.

Hierarchy Another common distinction is the one between *hierarchical* pooling and *global* pooling, with the latter often referred to as a separate process known as readout. Specifically, global pooling refers to methods that coarse a graph into a single node, effectively deleting all the topological information. A method is categorized as global pooling if it is fixed with $K = 1$, meaning it outputs a singular-node graph characterized solely by its attributes. Additionally, in global pooling, the connection function consistently maps to an empty set.

Conversely, all other methods are classified as hierarchical pooling operators. Hierarchical and global pooling operators can be both integrated within the same GNN architecture for graph-level tasks. Hierarchical pooling offers a multi-resolution view of the graph, enabling the GNN to progressively extract high-level properties. Global pooling, on the other hand, is used for generating graph embeddings that can be used as input of traditional layers designed for vector operations.

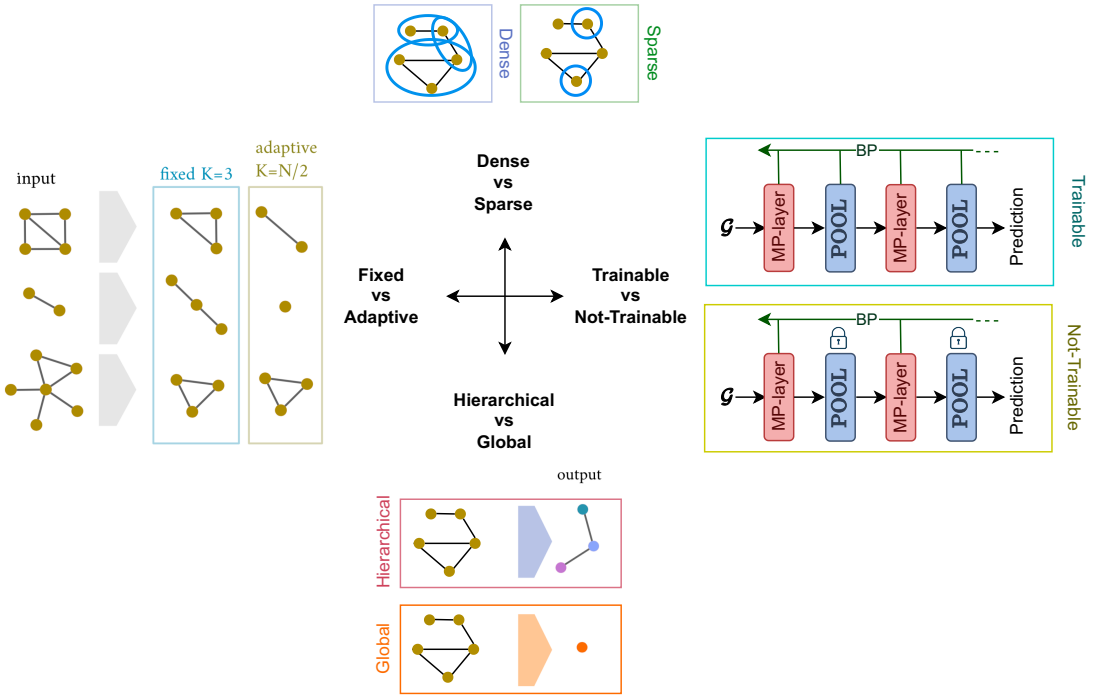


Figure 4.3: Taxonomy of pooling operators based on the SRC framework. Adapted from [1].

4.1.3 Existing Pooling Operators

In recent years, a variety of hierarchical pooling layers have been devised. Below, we delve into some of the most renowned.

Graclus The Graclus method [62], building on the principles of Metis [185], employs a greedy algorithm for graph coarsening, effectively minimizing spectral clustering objectives. It matches vertices based on a rule that maximizes the local normalized cut, leading to successive graph

reductions. In its pooling phase, Graclus halves the node count by clustering nodes based on edge weights or randomly, if no unclustered neighbors exist. During inference, max-pooling and fake vertices aid in maintaining a consistent reduction ratio, albeit introducing some noise into the graph representation. Graclus is not-trainable, sparse and adaptive.

DiffPool DiffPool, introduced in [64], represents the first effort in the development of end-to-end trainable pooling operators within GNNs. Unlike traditional pooling methods, DiffPool operates by training a GNN to derive a soft clustering matrix from the node features, enabling the aggregation of nodes into clusters. To ensure the creation of balanced clusters, DiffPool incorporates additional loss terms into its training, specifically targeting link prediction and entropy reduction. The regularization through entropy minimization facilitates the refinement of the cluster assignment, enhancing the overall robustness of the pooling process. By providing a differentiable framework for generating hierarchical representations of graphs, DiffPool offers versatility in its integration with various GNN architectures, enabling end-to-end learning. DiffPool is trainable, fixed and dense.

MinCut By approximating the minimum K-cut of the graph, MinCut [63] ensures balanced clustering while simultaneously optimizing task-specific objectives. At its core, a multi-layer perceptron (MLP) assigns nodes to clusters. This mechanism not only promotes the grouping of nodes with similar features — especially after message-passing layers that make the features of connected nodes similar — but also incorporates an unsupervised regularization loss. This loss function explicitly makes the MLP cluster together nodes that are strongly interconnected. The integration of this auxiliary loss enables MinCut to explicitly consider graph connectivity during the pooling process. Besides being trainable, MinCut is fixed and dense.

DMoN The DMoN pooling [186] aims to divide the graph into communities by maximizing the modularity. In the context of GNNs, DMoN employs a differentiable modularity score, allowing the network to learn how to assign nodes to communities directly as part of the training process. In particular, this is typically done by applying a softmax over predicted community affiliations for each node, resulting from the modularity optimization step. After assigning nodes to communities, DMoN aggregates the features of nodes within the same community summing or averaging them. During training, the network parameters are updated not just to perform well on the primary task (e.g., classification, regression) but also to improve the modularity of the learned community partitions. This joint optimization helps the network to learn meaningful community structures that are beneficial for the primary task. DMoN is trainable, fixed and dense.

Top-k The Top-k pooling approach, introduced in [181], learns a projection vector that is then applied to each node feature, resulting in the computation of a score for each node. Subsequently, the K nodes with the highest scores are retained, while the remaining nodes are discarded. This process facilitates memory efficiency by avoiding the need for generating cluster assignments. To address the non-differentiable nature of Top-k selection, the computed scores are used as a gate or attention mechanism for the node features, enabling the training of the projection vector through back-propagation. To ensure graph connectivity post-node removal, Top-k pooling

adjusts the adjacency matrix by dropping the corresponding rows and columns from A^2, albeit at a computational cost of $O(N^2)$. Top-k is trainable, adaptive and sparse.

SAGPool SAGPool [187] employs a self-attention mechanism with graph convolution to effectively discern which nodes should be retained or dropped, allowing for the learning of hierarchical representations in graphs in an end-to-end fashion. Unlike Top-k, the self-attention mechanism based on graph convolution enables SAGPool to consider both node features and graph topology, leading to a reduction in graph size that takes into account not only the node representations but also the graph structure. SAGPool is trainable, adaptive and sparse.

ASAPool ASAPool [188] operates by first identifying all potential local clusters within a given graph, utilizing a predefined receptive field. It then determines the nodes' cluster memberships through an attention mechanism. Specifically, a self-attention mechanism is employed to learn a cluster assignment matrix for each cluster, focusing on selecting relevant nodes to represent the cluster effectively. To do so, a new self-attention variant, named Master2Token (M2T), is introduced. The clusters are evaluated using a GNN, and a selection of the highest-scoring clusters is made to create the nodes of the pooled graph. In the final step, new edge weights are calculated for neighboring clusters, integrating both the cluster assignment matrix information and the original graph adjacency matrix. ASAPool is trainable, adaptive and sparse.

PANPool The PANpool strategy, as outlined in [182], leverages the Maximal Entropy Transition (MET) matrix, which reflects node importance through subgraph centrality metrics. This strategy does not require additional computations beyond those needed for convolution and adaptively identifies local motifs within the graph. By utilizing the MET matrix, PANPool effectively ranks nodes by their importance, facilitating adaptive graph coarsening without the need for predetermined centrality measures. This approach allows for data-driven, structure-aware graph reduction, maintaining the balance between local and global graph features. This pooling method is trainable, adaptive and sparse.

k-MIS The k-MIS pooling [180] works by coarsening the graph based on the concept of Maximal k-Independent Sets (k-MIS). In this method, the graph is reduced by selecting a set of nodes that are evenly spaced apart, ensuring no two selected nodes are within k hops of each other in the graph. This selection maintains the graph's structural integrity and reduces the size efficiently. The k-MIS forms the nodes of the reduced graph, and its edges are determined based on the original graph connectivity. This approach allows the preservation of topological properties and the graph's overall structure, making it suitable for tasks like graph classification and data compression while being implementable in parallel, optimizing for GPU acceleration. k-MIS is trainable, adaptive and sparse.

EdgePool The EdgePool method involves a distinctive edge contraction approach for pooling nodes. It starts by assigning scores to each edge using a trainable scoring function, i.e., $r_{ij} = f(x_i, x_j; \Theta)$. The algorithm then sequentially contracts edges, prioritizing those with higher scores. When two nodes i and j are connected by a high-scoring edge and are not already part of a contraction, EdgePool merges them into a new node. This process iteratively continues, with

the new node connecting to the original neighbors of the merged nodes. The EdgePool strategy of using edge-based pooling instead of node-based pooling is designed to retain essential graph structures and prevent the loss of any nodes. EdgePool is trainable, adaptive and sparse.

4.1.4 Evaluation of a Pooling Operator

Assessing the effectiveness of graph pooling methods involves multiple dimensions, as there is no universal metric. According to research outlined in [1], three distinct criteria are established to evaluate such operators, namely preserving the information content of the node attributes, preserving the topological structure and preserving the information required to solve various classification tasks. Specifically, for the first criterion, the ability to reconstruct the original coordinates of a geometric point cloud from its pooled version was evaluated. For the second criterion, the structural similarity between the original and coarsened graphs was analyzed by comparing the quadratic forms of their combinatorial Laplacian matrices. Lastly, for the third criterion, various graph classification benchmarks were considered; high classification accuracy suggests that a pooling operator can effectively preserve essential information tailored to the requirements of specific tasks. Experiments reveal that while no approach is universally superior, some methods excel under specific conditions. Not-trainable sparse strategies are preferred for preserving node characteristics while dense trainable solutions are more suitable for structural adherence. In terms of task-specific information, often central in machine learning, trainable methods are preferred, though no single approach is ideal for all scenarios. While not-trainable sparse methods, like Graclus, generally perform well, trainable dense methods, such as MinCut and DiffPool, provide greater adaptability and integration into GNN frameworks.

An unexplored avenue involves evaluating pooling operators in GNNs based on their ability to support graph information from a WL equivalence perspective. Given that pooling naturally incurs information loss, it becomes crucial to determine which operators can efficiently reduce the dimension of graphs without sacrificing the information essential for WL differentiation.

Upcoming sections will delve into both theoretical and empirical evaluations of pooling effectiveness in preserving or even enhancing the expressiveness of GNNs.

4.2 Pooling Operators can Preserve the Expressive Power of GNNs

Significant effort has been devoted to characterizing the expressive power of GNNs by comparing their capabilities with those of traditional algorithms for testing graph isomorphism. Despite the progress made in this direction, obtained results are still limited to flat GNNs consisting of a stack of MP layers followed by a final readout (Fig 4.4). While flat GNNs can at most match the computational power of the WL test, it is plausible that the graph coarsening process could lead to a loss of critical information, lowering this level of expressive power. Therefore, it is important to explore the specific conditions under which the pooling operation can be performed without causing graphs to lose necessary discriminating information. Additionally, despite the recent advances in the design of pooling operators, there is not a principled criterion to compare them. In the following subsections, we will present sufficient conditions for a pooling

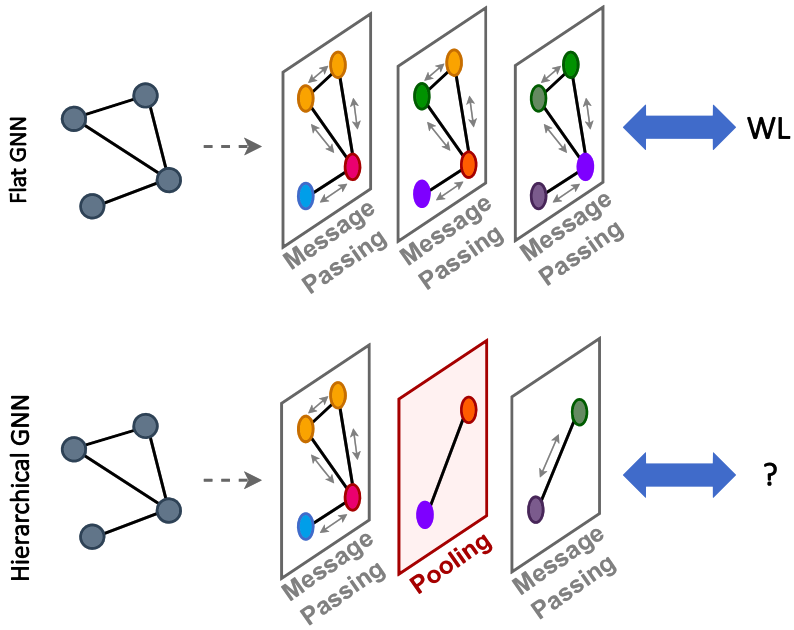


Figure 4.4: A flat GNN is at most as powerful as the WL test, while no results have been proven so far for hierarchical GNNs.

operator to fully preserve the expressive power of the MP layers before it. These conditions serve as a universal and theoretically grounded criterion for choosing among existing pooling operators or designing new ones. Based on our theoretical findings, we analyze several existing pooling operators and identify those that fail to satisfy the expressiveness conditions. Finally, we introduce an experimental setup to verify empirically the expressive power of a GNN equipped with pooling layers, in terms of its capability to perform a graph isomorphism test.

4.2.1 Conditions for Preserving the Expressive Power

The following theorem identifies three sufficient conditions that guarantee that a hierarchical GNN is as powerful as the WL test.

Theorem 2. *Let $G_1 = (V_1, E_1)$ with $|V_1| = N$ and $G_2 = (V_2, E_2)$ with $|V_2| = M$ with node features $\mathbf{X}$ and $\mathbf{Y}$ respectively, such that $G_1 \neq_{WL} G_2$. Let G_1^L and G_2^L be the graph obtained after applying a block of L MP layers such that $\mathbf{X}^L \in \mathbb{R}^{N \times F}$ and $\mathbf{Y}^L \in \mathbb{R}^{M \times F}$ are the new node features. Let* `POOL` *be a pooling operator expressed by the functions* `SEL`, `RED`, `CON`, *which is placed after the MP layers. Let $G_{1_P} =$ `POOL`(G_1^L) and $G_{2_P} =$ `POOL`(G_2^L) with $|V_{1_P}| = |V_{2_P}| = K$. Let $\mathbf{X}_P \in \mathbb{R}^{K \times F}$ and $\mathbf{Y}_P \in \mathbb{R}^{K \times F}$ be the node features of the pooled graphs so that the rows $\mathbf{x}_{P_j}$ and $\mathbf{y}_{P_j}$ represent the features of supernode j in graphs G_{1_P} and G_{2_P}, respectively. If the following conditions hold:*

1. $\sum_i^N \mathbf{x}_i^L \neq \sum_i^M \mathbf{y}_i^L$;

2. *The memberships generated by* SEL *satisfy* $\sum_{j=1}^{K} s_{ij} = \lambda$, *with* $\lambda > 0$ *for each node* i, *i.e., the cluster assignment matrix* $\boldsymbol{S}$ *is a right stochastic matrix up to the global constant* λ;

3. *The function* RED *is such that* $\texttt{RED} : (\mathbf{X}^L, \mathbf{S}) \mapsto \mathbf{X}_P = \mathbf{S}^T\mathbf{X}^L$;

then G_{1_P} *and* G_{2_P} *will have different node features, i.e., for all permutations of row indices* $\pi : \{1, \dots K\} \to \{1, \dots K\}$, $\mathbf{X}_P \neq \Pi(\mathbf{Y}_P)$, *where* $[\Pi(\mathbf{Y}_P)]_{ij} = \mathbf{y}_{P_{\pi(i)j}}$.

Proof. Let $\mathbf{S} \in \mathbb{R}^{N\times K}$ and $\mathbf{T} \in \mathbb{R}^{M\times K}$ be the matrices representing the cluster assignments generated by $\texttt{SEL}(G_1^L)$ and $\texttt{SEL}(G_2^L)$, respectively. When condition 2 holds, we have that the entries of matrices $\mathbf{S}$ and $\mathbf{T}$ satisfy $\sum_{j=1}^{k} s_{ij} = \lambda, \forall i = 1, \dots, N$, and $\sum_{j=1}^{K} t_{ij} = \lambda, \forall i = 1, \dots, M$. If condition 3 holds, then the j-th row of $\mathbf{X}_P$ is $\mathbf{x}_{P_j} = \sum_{i=1}^{N} \mathbf{x}_i^L \cdot s_{ij}$. The same holds for the j-th row of $\mathbf{Y}_P$, which is $\mathbf{y}_{P_j} = \sum_{i=1}^{M} \mathbf{y}_i^L \cdot t_{ij}$. Suppose that there exists a rows' permutation $\pi : \{1, \dots, K\} \to \{1, \dots, K\}$ such that $\mathbf{x}_{P_j} = \mathbf{y}_{P_{\pi(j)}}$ $\forall i = 1, \dots, M$, that is:

$$\sum_{i=1}^{N} \mathbf{x}_i^L \cdot s_{ij} = \sum_{i=1}^{M} \mathbf{y}_i^L \cdot t_{i\pi(j)}, \quad \forall j = 1, \dots, K,$$

which implies

$$\sum_{j=1}^{K}\sum_{i=1}^{N} \mathbf{x}_i^L \cdot s_{ij} = \sum_{j=1}^{K}\sum_{i=1}^{M} \mathbf{y}_i^L \cdot t_{i\pi(j)} \Leftrightarrow \sum_{i=1}^{N} \mathbf{x}_i^L \cdot \sum_{j=1}^{K} s_{ij} = \sum_{i=1}^{M} \mathbf{y}_i^L \cdot \sum_{j=1}^{K} t_{i\pi(j)} \overset{2}{\Leftrightarrow}$$
$$\overset{2}{\Leftrightarrow} \sum_{i=1}^{N} \mathbf{x}_i^L \cdot \lambda = \sum_{i=1}^{M} \mathbf{y}_i^L \cdot \lambda \Leftrightarrow \sum_{i=1}^{N} \mathbf{x}_i^L = \sum_{i=1}^{M} \mathbf{y}_i^L$$

contradicting condition 1. □

Note that there are no restrictions on the cardinality of the original sets of nodes, $|V_1| = N$ and $|V_2| = M$. Indeed, since the proof does not depend on the number of nodes in the original graphs, N can either be equal to M or not. Additionally, we only focus on pooled graphs with the same number of nodes, i.e., $|V_{1_P}| = |V_{2_P}| = K$. The case where $|V_{1_P}| \neq |V_{2_P}|$ is trivial since two graphs with different numbers of nodes are inherently not WL equivalent. A schematic summary of Theorem 2 can be found in Fig. 4.5.

Condition 1 is strictly related to the theory of multisets. Indeed, a major breakthrough in designing highly expressive MP layers was achieved by building upon the findings of Deep Sets [189]. Under the assumption that the node features originate from a countable universe, it has been formally proven that there exists a function that, when applied to the node features, makes the sum over a multiset of node features injective [28]. The universal approximation theorem guarantees that such a function can be implemented by an MLP. Moreover, if the pooling operator satisfies conditions 2 and 3, it will produce different sets of node features. Due to the injectiveness of the coloring function of the WL algorithm, two graphs with different multisets of node features will be classified as non-isomorphic by the WL test and, therefore, $G_{1_P} \neq_{\text{WL}} G_{2_P}$. This implies that the pooling operator effectively coarsens the graphs while retaining all the information necessary to distinguish them. Therefore, Theorem 2 guarantees that there is a specific choice of parameters for the MP layer which, when combined with a pooling operator that satisfies the conditions of the theorem, makes the resulting GNN architecture injective.

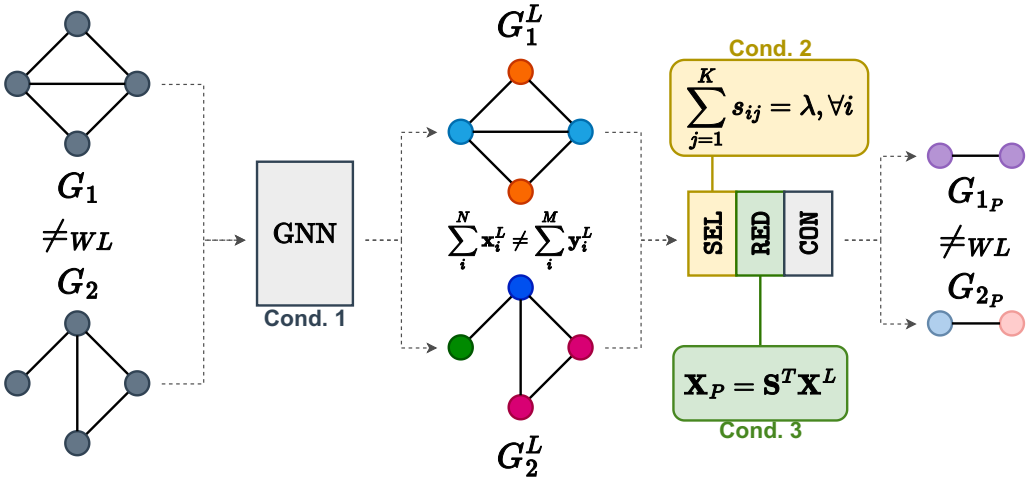


Figure 4.5: A GNN with expressive MP layers (condition 1) computes different features for two graphs G_1, G_2 that are WL-distinguishable. A pooling layer satisfying the conditions 2 and 3 generates coarsened graphs G_{1_P} and G_{2_P} that are still WL-distinguishable.

Condition 2 implies that *all* nodes in the original graph must contribute to the supernodes. Moreover, letting the sum of the memberships s_{ij} to be a constant λ (usually, $\lambda = 1$) places a restriction on the formation of the supernodes. Condition 3 requires that the features of the supernodes $\mathbf{X}_P$ are a convex combination of the node features $\mathbf{X}^L$. It is important to note that the conditions for the expressiveness only involve `SEL` and `RED`, but not the `CON` function. Indeed, both the graph topology and the node features are embedded in the features of the supernodes by MP and pooling layers satisfying the conditions of Th. 2. Nevertheless, even if a badly-behaved `CON` function does not affect the expressiveness of the pooling operator, it can still compromise the effectiveness of the MP layers that come afterward. This will be discussed further in Sections 4.2.3 and 4.2.4.

4.2.2 Expressiveness of Existing Pooling Operators

We claim that the expressiveness of existing pooling operators is closely related to whether they are dense or sparse. Specifically, all dense methods adhere to the conditions of Theorem 2, thereby maintaining expressiveness; sparse methods, on the other hand, can only be expressive if no graph nodes are discarded.

Dense pooling operators DiffPool, MinCutPool, and DMoN compute a cluster assignment matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$ either with an MLP or an MP-layer, which is fed with the node features $\mathbf{X}^L$ and ends with a `softmax`. The main difference among these methods is in how they define unsupervised auxiliary loss functions, which are used to inject a bias in how the clusters are formed. Thanks to the `softmax` normalization, the cluster assignments sum up to one, ensuring condition 2 of Th. 2 to be satisfied. Moreover, the pooled node features are computed as $\mathbf{X}_p = \mathbf{S}^{\mathrm{T}}\mathbf{X}^L$, making also condition 3 satisfied.

There are dense pooling operators that use algorithms such as non-negative matrix factorization [190] to obtain a cluster assignment matrix $\mathbf{S}$, which may not satisfy condition 2. Nonetheless, it is always possible to apply a suitable normalization to ensure that the rows in $\mathbf{S}$ sum up to a constant. Therefore, we claim that all dense methods preserve the expressive power of the preceding MP layers.

Non-expressive sparse pooling operators Top-k, ASAPool, SAGPool and PANPool reduce the graph by selecting a subset of its nodes based on a ranking score and mainly differ in how their `SEL` function computes such a score. Specifically, the Top-k method ranks nodes based on a score obtained by multiplying the node features with a trainable projection vector. A node i is kept ($s_i = 1$) if is among the Top-K in the ranking and is discarded ($s_i = 0$), otherwise. SAGPool simply replaces the projection vector with an MP layer to account for the graph structure when scoring the nodes. ASAPool, instead, examines all potential local clusters in the input graph given a fixed receptive field and employs an attention mechanism to compute the cluster membership of the nodes. The clusters are subsequently scored using a particular MP operator. Finally, in PANPool, cores are obtained from the diagonal entries of the maximal entropy transition matrix, which is a generalization of the graph Laplacian.

Regardless of how the score is computed, all these methods generate a cluster assignment matrix $\mathbf{S}$ where not all the rows sum to a constant. Indeed, if a node is not selected, it is not assigned to any supernode in the coarsened graph. Therefore, these methods fail to meet condition 2 of Theorem 2. Additionally, in the `RED` function of all these methods the features of each selected node are multiplied by its ranking score, making condition 3 also unsatisfied.

Intuitively, these operators produce a pooled graph that is a subgraph of the original graph and discard the content of the remaining parts. This hinders the ability to retain all the necessary information for preserving the expressiveness of the preceding MP layers. The limitation of Top-k is exemplified in Fig. 4.6: regardless of the projector p, Top-k maps two WL-distinguishable graphs into two isomorphic graphs, meaning that it cannot preserve the partition on graphs induced by the WL test.

Expressive sparse pooling operators Not all sparse pooling operators coarsen the graph by selecting a subgraph. In fact, some of them assign each node in the original graph to exactly one supernode and, thus, satisfy condition 2 of Th. 2. In matrix form and letting $\lambda = 1$, the cluster assignment would be represented by a sparse matrix $\mathbf{S}$ that satisfies $\mathbf{S}\mathbf{1}_K = \mathbf{1}_N$ and where every row has one entry equal to one and the others equal to zero. Within this category of sparse pooling operators, notable examples include Graclus, ECPool, and k-MISPool.

Graclus is a not-trainable, greedy bottom-up spectral clustering algorithm, which matches each vertex with the neighbor that is closest according to the graph connectivity [62]. When Graclus is used to perform graph pooling, the `RED` function is usually implemented as a `max_pool` operation between the vertices assigned to the same cluster [191]. To ensure condition 3 of Th. 2 to be satisfied, we use a `sum_pool` operation instead. Contrarily from Graclus, ECPool, and k-MISPool are trainable. ECPool first assigns to each edge $e_{i\to j}$ a score $r_{ij} = f(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\Theta})$. Then, iterates over each edge $e_{i\to j}$, starting from those with higher scores, and contracts it if neither nodes i and j are attached to an already contracted edge. The endpoints of a contracted edge are merged into a new supernode $\mathcal{S}_k = r_{ij}(\boldsymbol{x}_i + \boldsymbol{x}_j)$, while the remaining nodes become

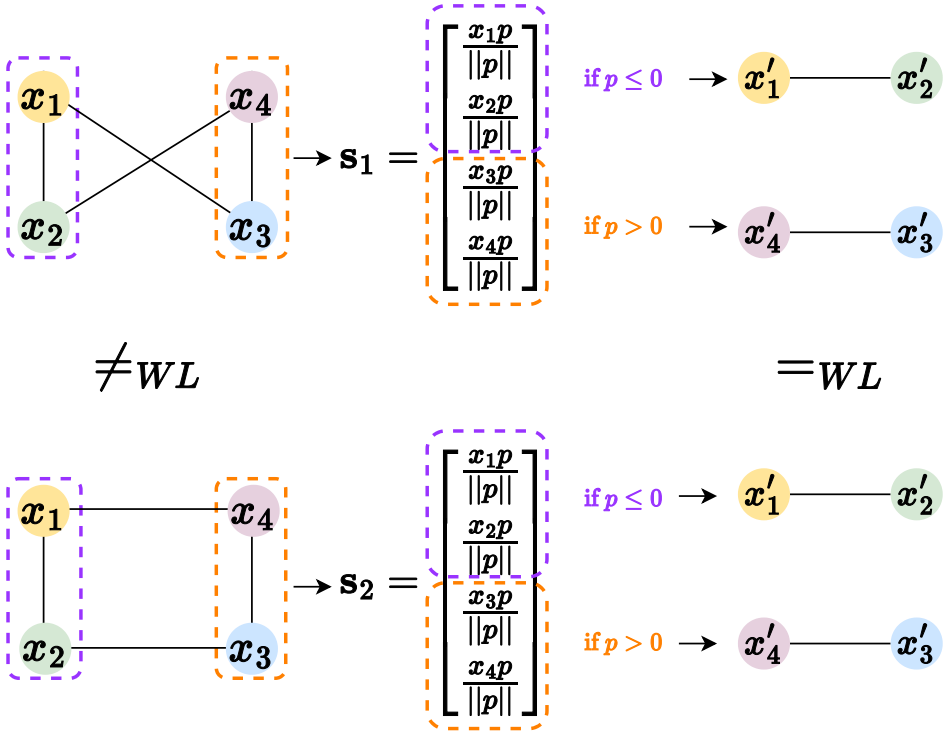


Figure 4.6: Example of failure of Top-k pooling. Given two WL-distinguishable graphs with node features $x_1 \leq x_2 \leq x_3 \leq x_4$, two scoring vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ are computed using a projector p. Then, the two nodes associated with the highest scores are selected. If $p \leq 0$, nodes 1 and 2 are chosen in both graphs. Conversely, if $p > 0$, nodes 3 and 4 are selected. Therefore, regardless of the value learned for the projector p, the two input graphs will be mapped into the same pooled graph.

supernodes themselves. Since each supernode either contains the nodes of a contracted edge or is a node from the original graph, all columns of $\mathbf{S}$ have either one or two entries equal to one, while each row sums up to one. The `RED` function can be expressed as $\mathbf{r} \odot \mathbf{S}^T \mathbf{X}^L$, where $\mathbf{r}[k] = r_{ij}$ if k is the contraction of two nodes i j, otherwise $\mathbf{r}[k] = 1$. Consequently, ECPool satisfies the expressiveness conditions of Th. 2. Finally, k-MISPool identifies the supernodes with the centroids of the maximal k-independent sets of a graph [192]. To speed up computation, the centroids are selected with a greedy approach based on a ranking vector π. Since π can be obtained from a trainable projector $\mathbf{p}$ applied to the vertex features, $\pi = \mathbf{X}^L \mathbf{p}^T$, k-MISPool is a trainable pooling operator. k-MISPool assigns each vertex to one of the centroids and aggregates the features of the vertex assigned to the same centroid with a `sum_pool` operation to create the features of the supernodes. Therefore, k-MISPool satisfies the expressiveness conditions of Th. 2.

A common characteristic of these methods is that the number of supernodes K cannot be directly specified. Graclus and ECPool achieve a pooling ratio of approximately 0.5 by roughly reducing each time the graph size by 50%. On the other hand, k-MISPool can control the coarsening level by computing the maximal independent set from G^k, which is the graph where

each node of G is connected to its k-hop neighbors. As the value of k increases, the pooling ratio decreases.

4.2.3 Criticism on Pooling

Recently, the effectiveness of graph pooling has been questioned using as an argument a set of empirical results aimed at exposing the weaknesses of certain pooling operators [193]. The experiments showed that using a randomized cluster assignment matrix $\mathbf{S}$ (followed by a `softmax` normalization) gives comparable results to using the assignment matrices learned by Diffpool [64] and MinCutPool [63]. Similarly, applying Graclus [62] on the complementary graph would give a performance similar to using the original graph.

We identify potential pitfalls in the proposed evaluation, which considered only pooling operators that are expressive and that, even after being modified, retain their expressive power. Clearly, even if expressiveness ensures that all the information is preserved in the pooled graph, its structure is corrupted when using a randomized $\mathbf{S}$ or a complementary graph. This hinders the effectiveness of the MP layers that come after pooling, as their inductive biases no longer match the data structure they receive. Notably, this might not affect certain classification tasks where the goal is to detect small structures, such as a motif in a molecule graph [133, 194], that are already captured by the MP layers before pooling.

To address these limitations, first, we propose to corrupt a pooling operator that is not expressive. In particular, we design a Top-k pooling operator where the nodes are ranked based on a score that is sampled from a normal distribution rather than being produced by a trainable function of the vertex features. Second, we evaluate all the modified pooling operators in a setting where the MP layers after pooling are essential for the task and show that the performance drop is significant.

4.2.4 Experimental Results

To empirically confirm the theoretical results presented in Section 4.2.1, we design a synthetic dataset that is specifically tailored to evaluate the expressive power of a GNN. We consider a GNN with MP layers interleaved with 10 different pooling operators: DiffPool [64], DMoN [186], MinCut [63], ECPool [195], Graclus, k-MISPool [180], Top-k [181], PANPool [182], ASAPool [188], and SAGPool [187]. For each pooling method, we use the implementation in Pytorch Geometric [196] with the default configuration. In addition, following the setup used to criticize the effectiveness of graph pooling [193], we consider the following pooling operators: Rand-Dense, a dense pooling operator where the cluster assignment is a normalized random matrix; Rand-Sparse, a sparse operator that ranks nodes based on a score sampled from a normal distribution; Cmp-Graclus, an operator that applies the Graclus algorithm on the complement graph.

The EXPWL1 dataset Our experiments aim at evaluating the expressive power of MP layers when combined with pooling layers. However, existing real-world and synthetic benchmark datasets are unsuitable for this purpose as they are not specifically designed to relate the power of GNNs to that of the WL test. Recently, the EXPool dataset was proposed to test the capability of special GNNs to achieve higher expressive power than the WL test [77], which, however, goes beyond the scope of our evaluation. Therefore, we introduce a modified version of EXPool,

called EXPWL1, which comprises a collection of graphs $\{G_1, \ldots, G_N, H_1, \ldots, H_N\}$ representing propositional formulas that can be satisfiable or unsatisfiable. Each pair (G_i, H_i) in EXPWL1 consists of two non-isomorphic graphs distinguishable by a WL test, which encode formulas with opposite SAT outcomes. Therefore, any GNN that has an expressive power equal to the WL test can distinguish them and achieve approximately 100% classification accuracy on the dataset. Compared to the original EXPool dataset, we increase the size of the dataset to a total of 3000 graphs and we also increase the size of each graph from an average of 55 nodes to 76 nodes. This was done to make it possible to apply an aggressive pooling without being left with a trivial graph structure. The EXPWL1 dataset and the code to reproduce the experimental results are publicly available[1].

In Figure 4.7, we report three graph pairs from the EXPWL1 dataset. In the figure, we use a different color map for each pair but the node features always assume a binary value in $\{0, 1\}$ in each graph.

Experimental procedure To empirically evaluate which pooling operator maintains the expressive power of the MP layers preceding it, we first identify a GNN architecture without pooling layers, which achieves approximately 100% accuracy on the EXPWL1 dataset. We tried configurations with a different number of GIN [28] layers followed by a `global_max_pool` or `global_sum_pool`. For the sake of comparison, we also consider a GNN with GCN layers [14], which are not expressive. The results are shown in Table 4.1.

Table 4.1: Performance of baseline architectures on EXPWL1.

MP layers	Global Pool	Test Acc
2 GIN	`global_max_pool`	$66.5_{\pm 1.8}$
2 GIN	`global_sum_pool`	$92.1_{\pm 1.0}$
2 GCN	`global_max_pool`	$62.3_{\pm 2.4}$
2 GCN	`global_sum_pool`	$76.7_{\pm 2.4}$
3 GIN	`global_max_pool`	$98.3_{\pm 0.6}$
3 GIN	`global_sum_pool`	$99.3_{\pm 0.3}$
3 GCN	`global_max_pool`	$97.4_{\pm 0.5}$
3 GCN	`global_sum_pool`	$98.7_{\pm 0.6}$

As expected, the architectures with GIN layers outperform those with GCN layers, especially when the layers are two. This is due to the fact that GCN implements mean pooling as an aggregator, which is a well-defined multiset function due to its permutation invariance, but it lacks injectiveness, leading to a loss of expressiveness. Similarly, GNNs with `global_sum_pool` perform better than those with `global_max_pool`, since the former is more expressive than the latter. An architecture with 3 GIN layers followed by `global_sum_pool` achieves approximately 100% accuracy on EXPWL1, making it the ideal baseline for our experimental evaluation. Perhaps more importantly, there is a significant difference in the performance of a 2-layer GNN followed by a global pooling layer that is more or less expressive. For this reason, the node embeddings generated by 2 GIN layers are a good candidate to test the expressiveness of the

[1] `https://github.com/FilippoMB/The-expressive-power-of-pooling-in-GNNs`

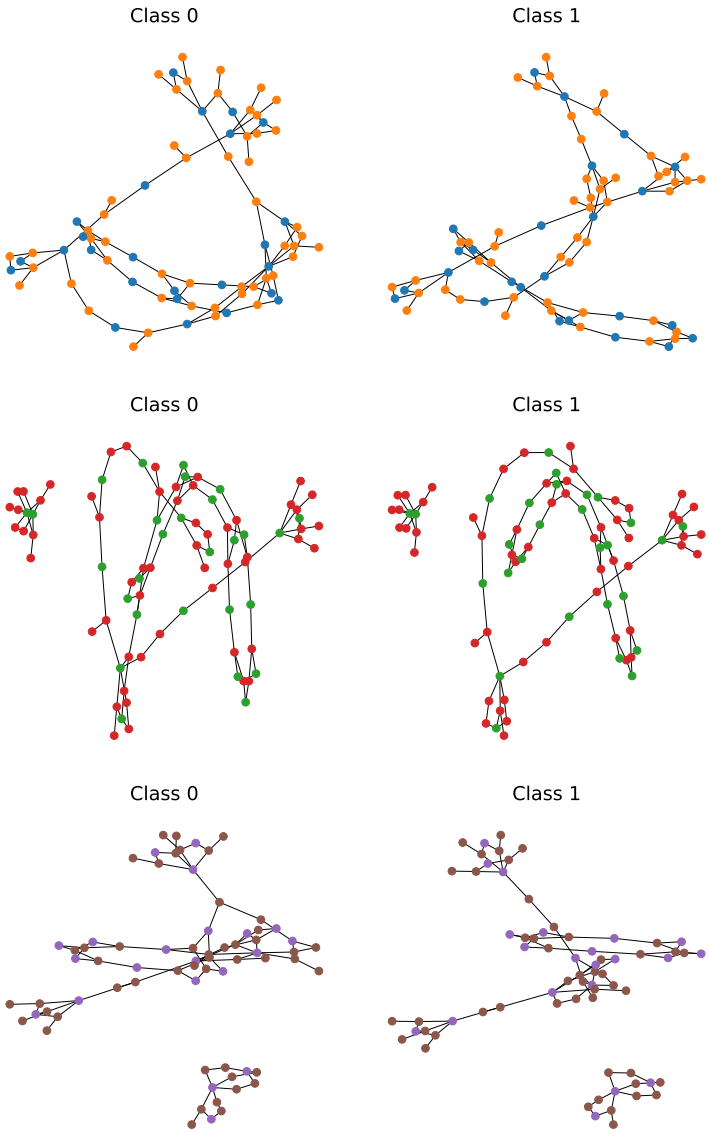


Figure 4.7: Three pairs of graphs from the EXPWL1 dataset. Each pair consists of two graphs with different classes that are WL distinguishable.

pooling operators considered in our analysis.
Then, we insert a pooling layer between the 2$^{\text{nd}}$ and 3$^{\text{rd}}$ GIN layer, which performs an aggressive pooling by using a pooling ratio of 0.1 that reduces the graph size by 90%. Each GIN layer is configured with an MLP with 2 hidden layers of 64 units and ELU activation functions. The readout is a 3-layer MLP with units [64, 64, 32], ELU activations, and dropout 0.5. The GNN is trained with Adam optimizer with an initial learning rate of 1e-4 using batches with size 32.

For SAGPool or ASAPool we us only one GIN layer before pooling. For PANPool we use 2 PanConv layers with filter size 2 instead of the first 2 GIN layers. The auxiliary losses in DiffPool, MinCutPool, and DMoN are added to the cross-entropy loss with weights [0.1,0.1], [0.5, 1.0], [0.3, 0.3, 0.3], respectively. For k-MIS we use $k = 5$ and we aggregate the features with the sum. Also for Graclus, we aggregate the node features with the sum.
To ensure a fair comparison, when testing each method, we shuffled the datasets and created 10 different training/validation/test splits using the same random seed. We trained each model on all splits for 500 epochs and reported the average training time and the average test accuracy obtained by the models that achieved the lowest loss on the validation set. To validate our experimental approach, we also measured the performance of the proposed GNN architecture equipped with the different pooling layers on popular benchmark datasets for graph classification [197, 198]. Table 4.2 reports the information about the datasets used in the experimental evaluation. Since the COLLAB and REDDIT-BINARY datasets lack vertex features, we assign a constant feature value of 1 to all vertices.

Table 4.2: Details of the graph classification datasets.

Dataset	**#Samples**	**#Classes**	**Avg. #vertices**
EXPWL1	3,000	2	76.96
NCI1	4,110	2	29.87
Proteins	1,113	2	39.06
COLORS-3	10,500	11	61.31
Mutagenicity	4,337	2	30.32
COLLAB	5,000	3	74.49
REDDIT-B	2,000	2	429.63
B-hard	1,800	3	148.32
MUTAG	188	2	17.93
PTC_MR	344	2	14.29
IMDB-B	1,000	2	19.77
IMDB-MULTI	1,500	3	13.00
ENZYMES	600	6	32.63
REDDIT-5K	4,999	5	508.52

Dataset	**Avg. #edges**	**Vertex attr.**	**Vertex labels**
EXPWL1	186.46	–	yes
NCI1	64.60	–	yes
Proteins	72.82	1	yes
COLORS-3	91.03	4	no
Mutagenicity	61.54	–	yes
COLLAB	4,914.43	–	no
REDDIT-B	995.51	–	no
B-hard	572.32	–	yes
MUTAG	19.79	–	yes
PTC_MR	14.69	–	yes
IMDB-B	96.53	–	no
IMDB-MULTI	65.94	–	no
ENZYMES	62.14	18	yes
REDDIT-5K	594.87	–	no

Results Table 4.3 reports the performance of different pooling operators on EXPWL1. These results are consistent with our theoretical findings: pooling operators that satisfy the conditions of Th. 2 achieve the highest average accuracy and, despite the aggressive pooling, they retain all the necessary information to achieve the same performance of a GNN without a pooling layer. On the other hand, non-expressive pooling operators achieve a significantly lower accuracy as they are not able to correctly distinguish all graphs.

Table 4.3: Classification results on EXPWL1.

Pooling	s/epoch	GIN layers	Pool Ratio	Test Acc	Expressive
No-pool	0.33s	3	–	$99.3_{\pm 0.3}$	✓
DiffPool	0.69s	2+1	0.1	$97.0_{\pm 2.4}$	✓
DMoN	0.75s	2+1	0.1	$99.0_{\pm 0.7}$	✓
MinCut	0.72s	2+1	0.1	$98.8_{\pm 0.4}$	✓
ECPool	20.71s	2+1	0.2	$100.0_{\pm 0.0}$	✓
Graclus	1.00s	2+1	0.1	$99.9_{\pm 0.1}$	✓
***k*-MIS**	1.17s	2+1	0.1	$99.9_{\pm 0.1}$	✓
Top-*k*	0.47s	2+1	0.1	$67.9_{\pm 13.9}$	✗
PanPool	3.82s	2+1	0.1	$63.2_{\pm 7.7}$	✗
ASAPool	1.11s	1+1	0.1	$83.5_{\pm 2.5}$	✗
SAGPool	0.59s	1+1	0.1	$79.5_{\pm 9.6}$	✗
Rand-dense	0.41s	2+1	0.1	$91.7_{\pm 1.3}$	✓
Cmp-Graclus	8.08s	2+1	0.1	$91.9_{\pm 1.2}$	✓
Rand-sparse	0.47s	2+1	0.1	$62.8_{\pm 1.8}$	✗

Table 4.3 also shows that employing a pooling operator based on a normalized random cluster assignment matrix (Rand-dense) or the complement graph (Cmp-Graclus) gives a lower performance. First of all, this result disproves the argument that such operators are comparable to the regular ones [193]. Additionally, we notice that the reduction in performance is less significant for Rand-Dense and Cmp-Graclus than for Rand-sparse. This outcome is expected because, in terms of expressiveness, Rand-dense and Cmp-Graclus still satisfy the conditions of Th. 2. Nevertheless, their performance is still lower than their regular counterparts. The reason is that, even if a badly-behaved `CON` function does not compromise the expressiveness of the pooling operator, the structure of the pooled graph is corrupted when utilizing a randomized $\mathbf{S}$ or a complementary graph. This, in turn, reduces the effectiveness of the last GIN layer, which is essential to correctly classify the graphs in EXPWL1.

There are two remarks about the experimental evaluation. As discussed in Section 4.1.3, it is not possible to explicitly specify the pooling ratio in Graclus, ECPool, and k-MISPool. For k-MISPool, setting $k = 5$ gives a pooling ratio of approximately 0.09 on EXPWL1. However, for Graclus, Cmp-Graclus, and ECPool, the only option is to apply the pooling operator recursively until the desired pooling ratio of 0.1 is reached. Unfortunately, this approach is demanding, both in terms of computing time and memory usage. While in EXPWL1 it was possible to do this for Graclus and Cmp-Graclus, we encounter an out-of-memory error after a few epochs when running ECPool on an RTX A6000 with 48GB of VRAM. Thus, the results for ECPool are obtained with a recursion that gives a pooling ratio of approximately 0.2. While this simplifies the training in ECPool we argue that, due to its expressiveness, ECPool would have reached approximately 100% accuracy on EXPWL1 if implementing a more aggressive pooling was feasible. The second

remark is that in EXPWL1 when using too many MP layers, at least one node ends up containing enough information to accurately classify the graphs. This is proved by the fact that a model with 3 GIN layers followed by `global_max_pool` achieves an accuracy of $98.3_{\pm 0.6}$ (Tab. 4.1). Note that the baseline model in Tab. 4.3 with 3 GIN layers equipped with the more expressive `global_sum_pool` achieves a slightly higher accuracy of $99.3_{\pm 0.3}$. In contrast, a model with only 2 GIN layers and `global_max_pool` gives a significantly lower accuracy of $66.5_{\pm 1.8}$. Therefore, to ensure that the evaluation is meaningful, no more than 2 MP layers should precede the pooling operator. Since ASAPool and SAGPool implement an additional MP operation internally, we use only 1 GIN layer before pooling, rather than 2 as for the other pooling methods. Finally, Fig. 4.8

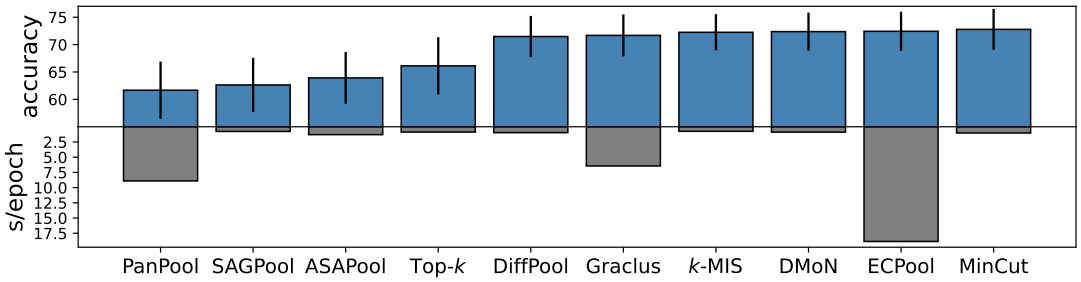


Figure 4.8: Average accuracy (and std.) v.s. average runtime on the benchmark datasets.

shows the average accuracy and the average run-time obtained on several benchmark datasets by a GNN equipped with the different pooling methods (the detailed results are in reported in Tabs. 4.4, 4.5). These benchmarks are not designed to test the expressive power and, thus, a GNN equipped with a non-expressive pooling operator could achieve good performance. Specifically, this happens in those datasets where all the necessary information is captured by the first two GIN layers that come before pooling or in datasets where only a small part of the graph is needed to determine the class. Nevertheless, this second experiment serves two purposes. First, it demonstrates the soundness of the GNN architecture used in the first experiment, which achieves results comparable to those of GNNs carefully optimized on the benchmark datasets [25]. Second, and most importantly, it shows that performance on the benchmark datasets and EXPWL1 are aligned; this underlines the relevance of our theoretical result on the expressiveness in practical applications. It is worth noting that on the benchmark datasets, it was not possible to obtain a pooling ratio of 0.1 for both Graclus and ECPool. Using a pooling ratio of 0.5 gives Graclus and ECPool an advantage over other methods, which makes the comparison not completely fair and shows an important limitation of these two methods.

As a concluding remark, we comment on the training time of the dense and sparse pooling methods. A popular argument in favor of sparse pooling methods is their computational advantage compared to the dense ones. Our results show that this is not the case in modern deep-learning pipelines. In fact, ECPool, Graclus, PANPool, and even ASAPool are slower than dense pooling methods, while the only sparse method with training times lower than the dense ones is k-MIS. Even if it is true that the sparse methods save memory by avoiding computing intermediate dense matrices, this is relevant only for very large graphs that are rarely encountered in most applications.

Table 4.4: Graph classification test accuracy on benchmark datasets.

Pooling	NCI1	PROTEINS	COLORS-3	Mutagenity	COLLAB
DiffPool	77.8±3.9	72.8±3.3	87.6±1.0	80.0±1.9	76.6±2.5
DMoN	78.5±1.4	73.1±4.6	88.4±1.4	81.3±0.3	80.9±0.7
MinCut	80.1±2.6	76.0±3.6	88.7±1.6	81.2±1.9	79.2±1.5
ECPool	79.8±3.3	69.5±5.9	81.4±3.3	82.0±1.6	80.9±1.4
Graclus	81.2±3.4	73.0±5.9	77.6±1.2	81.9±1.6	80.4±1.5
k-MIS	77.6±3.0	75.9±2.9	82.9±1.7	82.6±1.2	73.7±1.4
Top-k	72.6±3.1	73.2±2.7	57.4±2.5	74.4±4.7	77.9±2.1
PanPool	66.1±2.3	75.2±6.2	40.7±11.5	67.2±2.0	78.2±1.5
ASAPool	73.1±2.5	75.5±3.2	43.0±4.7	76.5±2.8	78.4±1.6
SAGPool	79.1±3.0	75.2±2.7	43.1±11.1	77.9±2.8	78.1±1.8
Rand-dense	78.2±2.0	75.3±1.3	83.3±0.9	81.4±1.8	69.3±1.6
Cmp-Graclus	77.8±1.8	73.6±4.7	84.7±0.9	80.7±1.8	OOM
Rand-sparse	69.1±3.3	74.6±4.2	35.5±1.1	69.8±1.0	68.8±1.6

Pooling	MUTAG	PTC_MR	IMDB-B	IMDB-MULTI
DiffPool	86.8±9.7	54.7±6.1	71.3±3.1	45.2±3.4
DMoN	86.3±7.1	55.5±7.3	71.9±3.3	47.0±5.5
MinCut	83.1±9.6	57.9±7.7	71.9±5.7	46.6±4.0
ECPool	90.0±7.2	54.7±8.4	71.3±3.4	49.2±2.9
Graclus	85.2±8.0	55.2±6.4	72.3±5.8	46.2±4.4
k-MIS	85.7±6.2	59.7±5.7	73.1±4.2	46.8±4.6
Top-k	78.4±11.8	58.2±8.9	70.9±3.3	44.8±2.9
PanPool	83.1±13.2	53.5±7.7	73.9±3.5	48.3±3.7
ASAPool	74.2±6.8	50.5±12.1	71.4±2.8	46.1±4.2
SAGPool	73.7±6.6	58.8±8.0	71.0±4.0	44.0±3.4
Rand-dense	88.9±4.3	56.1±9.7	70.5±3.4	45.2±5.6
Cmp-Graclus	83.2±9.1	55.9±4.6	OOM	OOM
Rand-sparse	68.9±17.3	56.4±5.9	71.6±3.6	45.8±3.7

Pooling	ENZYMES	REDDIT-5K	REDDIT-B	B-hard
DiffPool	62.3±7.3	53.7±1.8	89.9±2.8	70.2±1.5
DMoN	61.0±5.0	56.6±2.3	91.3±1.4	71.1±1.0
MinCut	62.3±3.8	56.2±2.8	91.9±1.8	71.2±1.1
ECPool	59.6±3.7	53.6±2.2	90.7±1.7	74.5±1.6
Graclus	61.0±6.6	52.3±1.4	92.9±1.7	72.3±1.3
k-MIS	63.5±7.1	56.4±2.3	90.6±1.4	71.7±0.9
Top-k	45.5±10.5	50.4±3.7	87.4±3.5	68.1±7.7
PanPool	40.5±5.0	46.5±2.4	83.6±1.9	44.2±8.5
ASAPool	44.8±7.6	48.8±1.6	88.0±5.6	67.5±6.1
SAGPool	41.6±5.2	49.9±2.9	84.5±4.4	54.0±6.6
Rand-dense	62.1±5.0	54.5±2.1	89.3±2.1	71.0±2.2
Cmp-Graclus	63.5±5.0	OOM	OOM	OOM
Rand-sparse	62.1±5.0	50.6±2.4	84.5±1.9	50.1±4.0

4.3 Pooling Operators can Increase the Expressive Power of GNNs

The results obtained so far suggest that a hierarchical GNN can be as expressive as the WL despite the information loss induced by pooling. A pertinent question that emerges is whether it

Table 4.5: Graph classification test run-time in s/epoch.

Pooling	NCI1	PROTEINS	COLORS-3	Mutagenity	COLLAB
DiffPool	0.83s	0.23s	1.67s	0.90s	1.68s
DMoN	1.01s	0.28s	1.94s	1.06s	1.83s
MinCut	0.95s	0.28s	1.82s	1.10s	1.82s
ECPool	4.39s	1.97s	10.30s	4.22s	44.11s
Graclus	0.95s	0.27s	2.47s	0.98s	3.01s
k**-MISPool**	0.88s	0.25s	2.48s	0.95s	1.38s
Top-k	1.04s	0.29s	2.78s	1.04s	2.79s
PanPool	2.81s	0.81s	7.16s	5.48s	7.67s
ASAPool	1.83s	0.52s	4.48s	1.80s	3.97s
SAGPool	1.09s	0.30s	2.52s	1.07s	2.81s
Rand-dense	0.54s	0.14s	1.44s	0.55s	0.88s
Cmp-Graclus	7.94s	3.27s	34.05s	1.94s	–
Rand-sparse	0.64s	0.18s	1.72s	0.68s	1.00s

Pooling	MUTAG	PTC_MR	IMDB-B	IMDB-MULTI
DiffPool	0.04s	0.07s	0.14s	0.28s
DMoN	0.05s	0.09s	0.17s	0.37s
MinCut	0.04s	0.08s	0.16s	0.35s
ECPool	0.08s	0.12s	1.66s	1.01s
Graclus	0.09s	0.14s	0.24s	0.33s
k**-MIS**	0.06s	0.10s	0.28s	0.21s
Top-k	0.04s	0.08s	0.24s	0.32s
PanPool	0.26s	0.49s	1.30s	1.92s
ASAPool	0.14s	0.10s	0.43s	0.40s
SAGPool	0.04s	0.05s	0.23s	0.27s
Rand-dense	0.04s	0.06s	0.13s	0.19s
Cmp-Graclus	0.17s	0.47s	–	–
Rand-sparse	0.04s	0.08s	0.15s	0.23

Pooling	ENZYMES	REDDIT-5K	REDDIT-B	B-hard
DiffPool	0.12s	4.52s	1.74s	0.29s
DMoN	0.15s	3.21s	1.04s	0.33s
MinCut	0.14s	4.45s	1.78s	0.35s
ECPool	0.54s	37.13s	3.17s	6.90s
Graclus	0.14s	74.02s	0.75s	0.31s
k**-MIS**	0.12s	2.02s	0.48s	0.43s
Top-k	0.13s	1.75s	0.47s	0.30s
PanPool	0.98s	34.3s	46.15s	6.27s
ASAPool	0.16s	1.89s	0.79s	0.52s
SAGPool	0.08s	1.09s	0.43s	0.28s
Rand-dense	0.12s	3.48s	1.44s	0.26s
Cmp-Graclus	1.38s	–	–	–
Rand-sparse	0.14s	1.67s	0.47s	0.31s

is feasible not only to preserve but also to enhance this expressive power through an appropriate pooling layer. Indeed, increasing the expressive power of GNNs to go beyond the limitations of WL is a significant and current research interest.

In recent years, significant efforts have been made to enhance the expressive power of GNNs. Several alternative GNN architectures have been proposed, such as k-GNN [199], which draws inspiration from the extension of the WL algorithm to k-tuples of nodes, or ESAN [75], which

encodes multisets of subgraphs instead of multisets of node features. Such expressive GNNs, however, usually result in a combinatorial explosion of the input data size.

In [67], we explore the potential of increasing the expressive power of GNNs while reducing the complexity and amount of computations and maintaining the standard message passing scheme. In particular, we show that pooling operators can produce coarsened graphs that are distinguishable by the WL test when applied to two non-isomorphic graphs that are indistinguishable by WL.

Guided by this theoretical framework, we develop a new hierarchical pooling method called XPool, specifically designed to meet sufficient conditions for enhancing the expressiveness of the GNN it is integrated into. Experiments on synthetic datasets, purposely crafted to test GNN expressiveness, show that our XPool pooling method can indeed increase the expressiveness of a GNN, even when it includes not-powerful message passing layers.

4.3.1 Conditions for Increasing the Expressive Power

A pooling operator increases expressive power if it ensures that any pair of non-isomorphic, WL-distinguishable graphs remain distinguishable after pooling and if there exists a pair of non-isomorphic WL-indistinguishable graphs that after the pooling become distinguishable. In the following theorems we will show that it is possible to construct a pooling operator that increases the expressive power.

Theorem 3. *Let* `POOL` *be a pooling operator expressed by the functions* `SEL`*,* `RED`*,* `CON`*, which is placed after the MP layers. If the following conditions hold:*

1. *The* `RED` *function is injective;*
2. *The* `SEL` *function is incomparable to or more expressive than WL test,*

then `POOL` *increases the expressive power.*

Proof. First of all, the injectivity of `RED` ensures that any pair of non-isomorphic, WL-distinguishable graphs remain distinguishable after pooling. Since `SEL` is incomparable to or more expressive than WL, by definition there exists a pair of graphs G_1 and G_2 that are not distinguishable by WL but can be distinguished by `SEL`. For these two graphs, $\texttt{SEL}(G_1) = \{S_1^1, \ldots, S_k^1\} \neq \{S_1^2, \ldots, S_l^2\} = \texttt{SEL}(G_2)$. Since `RED` is injective, independently of the `CON` function, the resulting coarsened graphs will be WL-distinguishable, i.e., $\texttt{POOL}(G_1) \neq_{\text{WL}} \texttt{POOL}(G_2)$. □

As an example of powerful `SEL`, consider a function that clusters together nodes that lie on the same cycle. This `SEL` function is incomparable to WL, as it cannot distinguish non-isomorphic trees of the same size. Figure 4.9 shows that a pooling operator with such a `SEL` function produces WL-distinguishable pooled graphs starting from two triangles from a cycle of length six which indeed are WL-indistinguishable. We point out that several existing methods benefit from this theoretical foundation, including CliquePool [200], CurvPool [201], and many others [202]. However, these methods utilize additional time-consuming graph algorithms to select nodes while also not being able to take node representations into account. For instance, CliquePool [200] relies on identifying cliques in a given graph, which has a non-polynomial worst-case runtime of $\mathcal{O}(3^{N/3})$ for N nodes [203].

In the following, we will thus show the possibility of constructing pooling operator that do not require expensive operations to increase the expressiveness of a GNN.

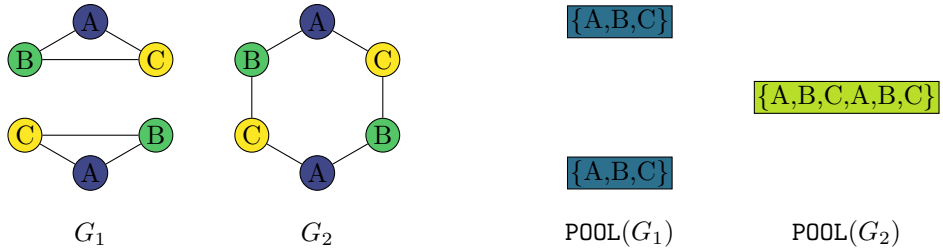


Figure 4.9: Two WL-indistinguishable graphs G_1, G_2 which can be distinguished after using a powerful SEL. Clustering cycles maps G_1 to a two supernode graph, while it maps G_2 to a single supernode graph.

Theorem 4. *Let* POOL *be a pooling operator expressed by the functions* SEL, RED, CON, *which is placed after the MP layers. If the following conditions hold:*

1. *The* RED *function is injective;*
2. *The* CON *function connects two supernodes if and only if they contain nodes that were connected in the original graph, i.e.,* $\texttt{CON}(G, \mathcal{S}) = \{(m, n) \mid \exists s_i^m, s_j^n > 0,\ (i, j) \in E\}$,

then, there exists a function SEL *such that, even though* $\texttt{SEL}(G_1) =_{WL} \texttt{SEL}(G_2)$ *for* $G_1 \neq_{WL} G_2$, *the resulting* POOL *increases the expressive power.*

Proof. First of all, the injectivity of RED ensures that any pair of non-isomorphic, WL-distinguishable graphs remain distinguishable after pooling. Then, we just need to find a pair of non-isomorphic but WL-indistinguishable graphs and a SEL function such that after pooling, the two graphs became distinguishable. First, let G_1 be composed of two disconnected triangles with the same set of node features but such that nodes within each triangle have distinct features. Let G_2, be a hexagon for which the nodes have the same three features of G_1 and the neighboring colors are the same for G_1 and G_2. This scenario is visualized in Figure 4.10. WL cannot distinguish these graphs. Now, let SEL be constructed such that we choose any pair of node features and contract all edges corresponding to that pair. For example, consider a boolean function $f : \mathbb{R}^F \times \mathbb{R}^F \to \{0, 1\}$ that decides whether a connected pair of nodes, given their F-dimensional features, get pooled together, as in their edge gets contracted or not. After this, we can define a MERGE function that repeatedly merges all overlapping node pairs $\{i, j\}, \{j, k\}$ to one single supernode $\{i, j, k\}$, while other nodes that do not get selected for a merge stay in their own supernode. Using the aforementioned construction of SEL and assuming CON to be as in Condition 2, we can observe that $\texttt{SEL}(G_1) = \texttt{SEL}(G_2)$ while $\texttt{POOL}(G_1) \neq_{WL} \texttt{POOL}(G_2)$. Indeed, for the triangle, this results in two disconnected edges. Conversely, for the hexagon, this results in a cycle of four nodes. □

Instead of relying on a complex additional algorithm, this insight provides the foundation for efficient and provably expressiveness-increasing pooling operators. The proof highlights the importance of including the node connectivities when forming supernodes. Transferring all edges between nodes in different supernodes to their respective supernode can result in a multigraph. We would not increase expressiveness by choosing CON to retain the number of individual edges between nodes from these supernodes, i.e., if we consider multigraphs. WL would not distinguish

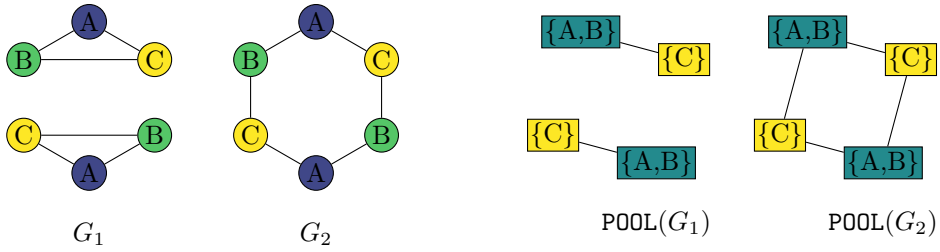


Figure 4.10: Two WL-indistinguishable graphs G_1, G_2 which can be distinguished after pooling. Contracting edges  and adding a single superedge between supernodes iff any original nodes were connected results in a disconnected graph for G_1 and a connected graph for G_2.

these multigraphs, and expressiveness would not be increased, as their resulting WL unfolding trees for two graphs would still be equal. On the other hand, pruning multiple edges to a single edge allows us to distinguish between graphs that WL could not distinguish. We attribute this to multiple edges between two supernodes corresponding to a cycle formed by their individual nodes. When deleting duplicate edges, we then detect that cycle. This allows us to distinguish this graph from other structurally similar graphs, which do not have that same cycle. Consequently, pooling methods considering the graph topology when clustering nodes have an advantage in expressiveness against pooling methods that take as input of the `SEL` function just the node features, such as DiffPool [64], DMoN [204].

4.3.2 XPool: An Expressive Pooling Operator

Guided by Theorem 3 and its constructive proof, we develop a novel pooling operator, XPool, which is able to enhance expressiveness of GNNs. Here, we delve into the specifics of the `SEL`, `RED` and `CON` functions implemented in XPool, providing a full explanation of their roles and mechanisms.

`SEL` First of all, we calculate a score for each edge of the graph as:

$$p_{i,j} = \text{MLP}(\mathbf{x}_i || \mathbf{x}_j),$$

where $\text{MLP} : \mathbb{R}^{2F} \to \mathbb{R}$, $\mathbf{x}_i$, $\mathbf{x}_j$ are the features of adjacent nodes i, j and $||$ denotes the concatenation. Subsequently, we select edges based on their scores, starting with the highest-scoring one. To ensure no edges are discarded, due to identical scores, we select all edges (i, j) with a score larger than the relative threshold:

$$\tau = \min_{i,j} p_{i,j} + \alpha \cdot (\max_{i,j} p_{i,j} - \min_{i,j} p_{i,j}) - \epsilon$$

where $\alpha \in [0, 1]$ denotes a hyperparameter. Here, $\epsilon > 0$ is a small constant, proportional to the machine precision, used to compare floating point representations. We then perform a merge operation on the selected node pairs. Below, we propose three levels of retaining the original graph structure.

- **MERGE** – We merge all pairs with overlapping nodes until the remaining pairs do not overlap. This results in the strongest loss of information in the graph, as any number of nodes could get merged into a single supernode.
- **SINGLE** – We only allow a single edge to be selected per node by iterating through the sorted edge scores and discard all node pairs that contain a repeated node.
- **MAX** – We only consider the edge with the highest score and discard all others. This corresponds to a minimal loss of information on the graph structure.

After performing one of these merge operations, unmerged nodes form additional supernodes, ensuring no graph nodes are excluded in the pooled graph creation. Each node contributes to only one supernode, resulting in a sparse cluster assignment matrix where each row features a single 1 — indicating node inclusion in a supernode — and all other entries are set to zero.

`RED` Following [28], we choose a sufficiently powerful `RED` consisting of two learnable functions $\phi : \mathbb{R}^F \to \mathbb{R}^F$ and $\psi : \mathbb{R}^F \to \mathbb{R}^F$ that make sum aggregation injective:

$$\texttt{RED} : (\mathcal{S}, \mathbf{X})_j = \psi \left(\sum_{i=1}^{n} s_i^j \cdot \phi(\mathbf{x}_i) \right).$$

In pratice, ϕ and ψ can be approximated through two MLPs.

`CON` Following condition 2 of Theorem 4, we set `CON` to connect two supernodes if and only if they contain nodes that were connected in the original graph, i.e.,

$$\texttt{CON}(G, \mathcal{S}) = \{(m, n) \mid \exists s_i^m, s_j^n > 0,\ (i, j) \in E\}.$$

4.3.3 Experimental Results

To validate our theoretical findings and evaluate the capabilities of XPool operator, we conduct a series of experiments. We integrate the XPool operator into three different GNNs and compare their empirical expressiveness against existing pooling operators.

Experimental Setup We evaluate the empirical expressiveness using the Brec framework [205]. The framework takes pairs of WL indistinguishable graphs, creates multiple permutations of both graphs, and is optimized to map non-isomorphic graphs to different embeddings using contrastive training. The framework computes a distance score between embeddings and counts each pair as distinguished if the score is larger than a predefined threshold. If a pair is distinguished, the framework also requires embeddings of additional permutations of the same graph to be sufficiently close. Otherwise, the sample is considered to be unreliable and the total score is not increased. For each considered method and dataset, we present the total number of distinguished and stable graph pairs.

All models use a uniform GNN structure to ensure consistency across different implementations. The initial feature of each node is determined by its degree. We then employ a linear layer followed by a ReLU activation function as a feature encoder. The core of the GNN consists of

a sequence of four message passing layers and one pooling layer repeated twice plus four other message passing layers. We use three different message passing layers, namely GCN [14], GAT [16] and GIN [28]. After the convolutional and pooling stages, a global sum pooling aggregates the node features, followed by a two-layer MLP. We use two datasets explicitly tailored to evaluate the expressiveness of GNNs, namely the BREC [205] and the RIGID [206] datasets.

BREC comprises 400 pairs of non-isomorphic graphs divided into four main categories: Basic, Regular, Extension, and CFI graphs. Basic Graphs include 60 pairs of 10-node graphs, Regular Graphs consist of 140 pairs, including simple and strongly regular graphs, Extension Graphs are made up of 100 pairs inspired by GNN extensions, and CFI Graphs, the most complex category, includes 100 pairs based on the Cai-Fürer-Immerman method [207]. BREC stands out for its range of graph difficulties, different assessments of GNNs, and large scale, making it a significant benchmark for studies in GNN expressiveness.

RIGID consists of 1086 pairs of graphs and was originally devised as a benchmark set of difficult instances for graph isomorphism algorithms that are WL-indistinguishable. The graphs, called shrunken multipedes, are unlikely to have nontrivial automorphisms and are constructed by transformations of a rigid bipartite multipede base graph [208] using the gadgets of Cai et al.[207]. There are six subsets of cfi-rigid pairs generated for different parameters. To evaluate many different pooling operators and graph convolutions, we consider all pairs for which each graph has at most 2000 nodes. This leads to a total amount of 598 graph pairs divided into 108 (r2), 190 (t2), 88 (z2), 124 (s2), 44 (z3), and 44 (d3) graph pairs.

Results The mean number of distinguished graph pairs and the standard deviation over three runs are displayed in Table 4.6. The GNN without pooling layer achieve zero correctly distinguished graphs for the BREC dataset and three correctly distinguished graphs for the RIGID dataset. This is due to numerical instabilities with floating points, which also initially motivated the construction of the BREC dataset [205]. For both the BREC and the RIGID dataset, our XPool significantly increases expressiveness in all the considered GNN models. Improvements are up to ten times over the second best-performing pooling method. For the BREC dataset, EdgePool and ASAPool are the only two baseline methods to achieve a non-zero number of distinguished graphs. For the RIGID dataset, all pooling methods improve slightly over the models without pooling.

We also observe large differences between our three proposed versions for merging selected edges. XPool (MERGE) achieves the lowest scores in all cases, highlighting that the potential gains in expressiveness are strongly reduced when combining many nodes simultaneously. In contrast, XPool (MAX) only pools a single edge per graph, leading to significantly improved results on the BREC dataset as the largest graphs contain only 198 nodes. Our considered graphs in the RIGID dataset contain up to 2000 nodes, for which pooling a single edge is less crucial. Going forward, we thus argue that XPool (SINGLE) serves as a desirable trade-off.

Table 4.6: Number of graph pairs correctly distinguished by each GNN using different pooling operators. Mean and standard deviation over three runs are shown.

Pooling Methods	GCN	GAT	GIN
BREC dataset (400 pairs)			
No Pooling	0 ± 0	0 ± 0	0 ± 0
EdgePool	15 ± 4	10 ± 2	17 ± 0
DiffPool	0 ± 0	0 ± 0	0 ± 0
Top-k	0 ± 0	0 ± 0	0 ± 0
SAGPool	0 ± 0	0 ± 0	0 ± 0
ASAPool	6 ± 0	6 ± 0	8 ± 2
XPool (MERGE)	22 ± 1	20 ± 1	22 ± 1
XPool (SINGLE)	52 ± 4	21 ± 1	62 ± 2
XPool (MAX)	$\mathbf{101} \pm 1$	$\mathbf{97} \pm 1$	$\mathbf{112} \pm 5$
RIGID dataset (598 pairs)			
No Pooling	3 ± 0	3 ± 0	3 ± 0
EdgePool	25 ± 2	36 ± 2	30 ± 4
DiffPool	3 ± 0	4 ± 1	3 ± 0
Top-k	23 ± 2	21 ± 8	36 ± 9
SAGPool	67 ± 3	71 ± 6	36 ± 4
XPool (MERGE)	8 ± 2	7 ± 1	2 ± 0
XPool (SINGLE)	$\mathbf{267} \pm 1$	$\mathbf{213} \pm 8$	$\mathbf{386} \pm 5$
XPool (MAX)	18 ± 2	15 ± 1	43 ± 2

4.4 Conclusions

In conclusion, both theoretical analysis and experimental evidence have demonstrated that the pooling layer is crucial for the expressiveness of the graph neural network in which it is incorporated. This thesis has outlined the sufficient conditions for preserving the expressiveness of message passing before pooling, as well as those for enhancing it; this opens up avenues for further research in this area to identify additional sufficient conditions and, more broadly, to design efficient pooling layers that aim not only at dimensionality reduction but also at enhancing expressive power.

Chapter 5

Conclusion

In this thesis, we presented and analyzed both theoretical and experimental findings related to GNNs, focusing on two main aspects: (i) the analysis of GNN-based models designed to handle temporal graphs, and (ii) the study of pooling layers for dimensionality reduction within GNN architectures.

Regarding (i), we proposed a systematic formalization of tasks and learning settings for TGNNs and a comprehensive taxonomy categorizing existing methods and highlighting unaddressed tasks. Building on this systematization of the current state-of-the-art, we discussed open challenges that needed to be addressed to unleash the full potential of TGNNs. We stressed the fact that the issues open to date were very challenging, since they presupposed considering both the temporal and relational dimensions of data, suggesting that forthcoming new computational models must go beyond the GNN framework to provide substantially better solutions. We concluded by tackling one of the identified open challenges: analyzing the expressive power of TGNNs. We focused on a specific category within our taxonomy, the STGNNs, which were among the most utilized and intuitive GNN-based models for temporal graphs. We proved that STGNNs were able to approximate, in probability and up to any degree of precision, any functions that preserve the equivalence induced by a specific dynamic version of the WL test, which we defined. Finally, we validated our approximation theorem through a carefully designed experimental setup.

Concerning (ii), we first identified sufficient conditions that a pooling operator must satisfy to fully preserve the expressive power of the original GNN model. Based on our theoretical results, we proposed a principled approach to evaluate the expressive power of existing graph pooling operators by verifying whether they met the conditions for expressiveness. To empirically test the expressive power of a GNN, we introduced a new dataset that allowed understanding whether a GNN architecture achieved the same discriminative power as the WL test. We used such a dataset to evaluate the expressiveness of a GNN equipped with different pooling operators and found that the experimental results were consistent with our theoretical findings. In our experimental evaluation, we also considered popular benchmark datasets for graph classification and found that the expressive pooling operators achieved higher performance. This confirmed the relevance in practical applications of our principled criterion to select a pooling operator based on its expressiveness. Finally, we showed that it was possible to go further, i.e., to construct hierarchical pooling operators that increased the expressiveness of GNNs while reducing the size of the computational graph. While all the popular proposed architectures capable of going beyond the limit imposed by WL were based on modifications of message passing—typically incurring substantial computational costs—we presented the first attempt to increase the expressive power of GNNs by acting on the hierarchical pooling mechanism. In particular, we identified sufficient conditions on pooling operators that allowed a hierarchical GNN to distinguish non-isomorphic

graphs that were indistinguishable by the WL test. Since no currently existing pooling method was specifically designed to meet the identified conditions, we designed a practical pooling operator that increased expressiveness. Our empirical evaluation confirmed theoretical findings, demonstrating that computationally light pooling functions could lead to more expressive GNNs.

Several potential directions for future research emerge from this thesis. A primary avenue is the further comparison of existing models for temporal graphs by testing them against a unified benchmark, such as the promising TGB framework. A more comprehensive understanding could also be achieved by considering models specifically designed for particular types of temporal graphs, such as spatio-temporal graphs [95, 209, 176] or strictly evolving graphs [210]. Moreover, addressing additional open challenges outlined in this thesis would be of interest, especially the development of tailored TGNN models for high-interest problems like weather forecasting or climate modeling, where the application of GNN-based models could yield precise and valuable solutions.

Concerning the study of pooling in GNNs, an interesting future research direction involves exploring the relationship between pooling layers and other GNN learning challenges, such as oversmoothing [211] and oversquashing [212]. Specially designed pooling techniques could offer a solution to these issues [201]. Additionally, the potential of using pooling to achieve structural embedding [213] offers an opportunity to address expressiveness challenges in link prediction tasks [214]. Finally, another promising research path could involve merging the two main themes of this thesis: exploring the construction of temporal-specific pooling designed explicitly for GNNs operating on temporal graphs.

Appendix A

Appendix

In this chapter, we present the formal proof of Thm 1. Prior to this, it is necessary to introduce some preliminary definitions and lemmas.

Definition A.0.1 (Unfolding Tree)**.** *The unfolding tree* $\mathbf{T}_v^d$ *in graph* $G_1 = (V_1, E_1, \mathbf{X}_1^V, \mathbf{X}_1^E)$ *of node* $v \in V$ *up to depth* $d \in \mathbb{N}_0$ *is defined as*

$$\mathbf{T}_v^d := \begin{cases} Tree(\mathbf{x}^v), & \text{if } d = 0 \\ Tree\big(\mathbf{x}^v, \{\mathbf{T}_u^{d-1}\}_{u \in \mathcal{N}[v]}\big) & \text{if } d > 0, \end{cases}$$

where $Tree(\mathbf{x}_v)$ *is a tree constituted of node* v *with attribute* $\mathbf{x}_v$*.* $Tree\big(\mathbf{x}^v, \{\mathbf{T}_u^{d-1})\}_{u \in \mathcal{N}[v]}\big)$ *is the tree of depth* d *consisting of the root node* v *connected to trees of depth* $d-1$ *rooted at each neighbor* u *of* v*, namely* $\mathbf{T}_u^{d-1}$*. We call unfolding tree of* v $\mathbf{T}_v := \lim_{d \to \infty} \mathbf{T}_v^d$*.*

Definition A.0.2 (Unfolding Equivalence)**.** *Let* $G_1 = (V_1, E_1, \mathbf{X}_1^V, \mathbf{X}_1^E)$ *and* $G_2 = G_2 = (V_2, E_2, \mathbf{X}_2^V, \mathbf{X}_2^E)$ *be two graphs.* G_1 *and* G_2 *are unfolding tree equivalent, noted by* $G_1 =_{UT} G_2$*, iff* $\{\mathbf{T}_u \mid u \in \mathcal{V}_1\} = \{\mathbf{T}_v \mid v \in \mathcal{V}_2\}$*. Analogously, two nodes* $u \in V_1, v \in V_2$ *are unfolding tree equivalent, noted by* $u =_{UT} v$ *iff* $\mathbf{T}_u = \mathbf{T}_v$*.*

Definition A.0.3 (Dynamic Unfolding Tree)**.** *Let* $G_T^S = \{G_i : i = 1, \ldots, n\}$ *be an STG as defined in Def. 3.1.3, with* V *the set of all nodes that appear at least in one timestamp. The dynamic unfolding tree* $\mathbf{DT}_v^d$ *of node* $v \in V$ *up to depth* $d \in \mathbb{N}_0$ *is a sequence of unfolding trees* $\mathbf{DT}_v^d := [\mathbf{T}_v^d(t_i)]_{i=1,\ldots,n}$*. If* $v \in V$*, then* $\mathbf{T}_v^d(t_i)$ *is defined in the following:*

$$\mathbf{T}_v^d(t_i) := \begin{cases} Tree(\mathbf{x}^v), & \text{if } d = 0 \\ Tree\big(\mathbf{x}^v, \{\mathbf{T}_u^{d-1}(t_i))\}_{u \in \mathcal{N}_{t_i}[v]}\big) & \text{if } d > 0, \end{cases}$$

where $Tree(\mathbf{x}_v)$ *is a tree constituted of node* v *with attribute* $\mathbf{x}_v$*.* $Tree\big(\mathbf{x}^v, \{\mathbf{T}_u^{d-1}(t_i))\}_{u \in \mathcal{N}_{t_i}[v]}\big)$ *is the tree of depth* d *consisting of the root node* v *connected to trees of depth* $d-1$ *rooted at each neighbor* u *of* v*, namely* $\mathbf{T}_u^{d-1}(t_i)$*.*
If $(v, \mathbf{x}^v) \notin V_i$*, then* $\mathbf{T}_v^d(t_i) := \mathbf{T}^\perp$ *for every* d*. We call dynamic unfolding tree of* v $\mathbf{DT}_v := \lim_{d \to \infty} \mathbf{DT}_v^d$*.*

Definition A.0.4 (Dynamic Unfolding Equivalence)**.** *Two nodes* $u,\ v \in V$ *are said to be dynamic unfolding equivalent* $u =_{DUT} v$ *iff* $\mathbf{DT}_u = \mathbf{DT}_v$*, that is* $\mathbf{T}_u(t_i) = \mathbf{T}_v(t_i)$ *for every timestep* t_i*. Analogously, two STGs* G_1*,* G_2 *are said to be dynamic unfolding equivalent* $G_1 =_{DUT} G_2$*, iff* $\{\mathbf{DT}_u \mid u \in \mathcal{V}_1\} = \{\mathbf{DT}_v \mid v \in \mathcal{V}_2\}$*.*

Theorem 5. *Let* G_T^S *be an STG with* V *the total set of nodes. Then, it holds*

$$\forall\, u, v \in V, u =_{DUT} v \iff u =_{DWL} v.$$

Proof. By Definition 3.7.2 , two nodes $u, v \in V$ are DWL equivalent iff they are WL equivalent at each timestamp, i.e.,

$$u =_{DWL} v \iff u =_{WL} v \quad \forall t_i.$$

The same holds for the DUT equivalence A.0.4, i.e.,

$$u =_{DUT} v \iff u =_{UT} v \quad \forall t_i.$$

Thus, in order to prove that $u =_{DUT} v \iff u =_{DWL} v$, it is sufficient to prove

$$u =_{UT} v \iff u =_{WL} v,$$

or, analogously, that

$$\forall\, d \in \mathbb{N}_0 : \mathbf{T}_u^d = \mathbf{T}_v^d \iff c_u^d = c_v^d. \tag{A.1}$$

The proof is carried out by induction on d, which represents both the depth of the unfolding trees and the iteration step in the WL coloring.

$\underline{d = 0}$: It holds

$$\mathbf{T}_u^0 = Tree(\mathbf{x}^u) = Tree(\mathbf{x}^v) = \mathbf{T}_v^0$$
$$\iff \mathbf{x}^u = \mathbf{x}^v \iff c_u^0 = \mathrm{HASH}(\mathbf{x}^u) = \mathrm{HASH}(\mathbf{x}^v) = c_v^0.$$

$\underline{d > 0}$: Suppose that Eq. A.1 holds for $d-1$, and prove that it holds also for d.

- By definition, $\mathbf{T}_u^d = \mathbf{T}_v^d$ is equivalent to

$$\begin{cases} \mathbf{T}_u^{d-1} = \mathbf{T}_v^{d-1} \quad \text{and} \\ \mathrm{Tree}\big(\mathbf{x}^u, \{\mathbf{T}_n^{d-1}\}_{n\in\mathcal{N}(u)}\big) = \mathrm{Tree}\big(\mathbf{x}^v, \{\mathbf{T}_m^{d-1}\}_{m\in\mathcal{N}(v)}\big). \end{cases} \tag{A.2}$$

- Applying the induction hypothesis, it holds that

$$\mathbf{T}_u^{d-1} = \mathbf{T}_v^{d-1} \iff c_u^{d-1} = c_v^{d-1}. \tag{A.3}$$

- From Eq. (A.2)

$$\{\mathbf{T}_n^{d-1}\}_{n\in\mathcal{N}(u)} = \{\mathbf{T}_m^{d-1}\}_{m\in\mathcal{N}(v)} \tag{A.4}$$

By the induction hypothesis, Eq. (A.4) is equivalent to

$$\{c_n^{d-1}\}_{n\in\mathcal{N}(u)} = \{c_m^{d-1}\}_{m\in\mathcal{N}(v)} \tag{A.5}$$

- Putting together Eq. (A.3), (A.5), and the fact that the HASH function is injective, we obtain:

$$c_u^d = \texttt{HASH}(c_u^{d-1}, \{c_n^{d-1} | n \in \mathcal{N}[u]\}) = \texttt{HASH}(c_v^{d-1}, \{c_m^{d-1} | m \in \mathcal{N}[v]\}) = c_v^d$$

□

Thanks to Thm. 5, we can interchangeably use the DWL equivalence and the DUT equivalence. Specifically, a dynamic system $dyn(\cdot,\cdot,\cdot) \in \mathcal{F}(\mathcal{G}_\mathcal{T})$ if and only if it preserves the DUT equivalence. Henceforth, in all subsequent theorems and proofs, we will use the DUT equivalence.

Theorem 6 (Functions of dynamic unfolding trees)**.** *A dynamic system dyn belongs to $\mathcal{F}(\mathcal{G}_\mathcal{T})$ iff there exists a function κ defined on trees such that for all the triplets (t_j, G_T^S, v), it holds:*

$$dyn(t_j, G_T^S, v) = \kappa\Big([\mathbf{T}_v(t_i)]_{i=1,...,j}\Big),$$

where $[\mathbf{T}_v(t_i)]_{i=1,...,j}$ is the sequence of unfolding threes up to time t_j in which the dynamical system is calculated.

Proof. We show the proposition by proving both directions of the equivalence relation:

$\Leftarrow$: This direction is trivial. Indeed, if $dyn(t_j, G_T^S, v) = dyn(t_j, G_T^S, u)$ for $u =_{DUT} v$ then

$$\kappa\Big([\mathbf{T}_v(t_i)]_{i=1,...,j}\Big) := dyn(t_j, G_T^S, v)$$

is well defined.

$\Rightarrow$: Suppose there exists κ such that $dyn(t_j, G_T^S, v) = \kappa\Big([\mathbf{T}_v(t_i)]_{i=1,...,j}\Big)$. Then, for any pair of nodes $u, v \in V$ with $u =_{DUT} v$ it holds:

$$dyn(t_j, G_T^S, v) = \kappa\Big([\mathbf{T}_v(t_i)]_{i=1,...,j}\Big) = \kappa\Big([\mathbf{T}_u(t_i)]_{i=1,...,j}\Big) = dyn(t_j, G_T^S, u).$$

□

Now we are ready to prove Thm. 1. Because of [215, Lem. 1], [215, Lem. 2],[216] and Thm 5, Thm. 1 is equivalent to the following theorem, where the domain contains a finite number of elements in $\mathcal{G}_\mathcal{T}$ and the attributes are integers. Hence, proving this theorem is sufficient to prove Thm. 1.

Theorem 7. *Given*

- $\mathcal{G}_\mathcal{T} = \{(t_i^p, G_T^{Sp}, v^p), p = 1, ..., P\}$*, with* $N = \max_{G_T^S \in \mathcal{G}_\mathcal{T}} |G_T^S|$ *and with integer features;*
- $dyn(\cdot, \cdot, \cdot) \in \mathcal{F}(\mathcal{G}_\mathcal{T})$ *measurable;*
- $\epsilon > 0$*;*

then, there exists an STGNN with $2N - 1$ layers such that:

$$||dyn(t_i^p, G_T^{Sp}, v^p) - STGNN(t_i^p, G_T^{Sp}, v^p)|| \leq \epsilon \quad \forall p = 1, ..., P \quad \forall t_i. \tag{A.6}$$

Proof. The proof involves assuming that the output dimension is $m = 1$, but the result can be extended to the general case with $m \in \mathbb{N}$ by concatenating the corresponding results. As a result of Thm. 6, there exists a function κ, s.t. $dyn(t_i^p, G_T^{Sp}, v^p) = \kappa\Big([\mathbf{T}_v(t_i)]_{i=1,...,j}\Big)$, $\forall p = 1, ..., P$ and $\forall t_i$. For Theorem 4.1.3 in [217], in order to store the information of a snapshot graph at time t_i with N_{t_i} nodes, an unfolding tree of depth $2N_{t_i} - 1$ is sufficient, i.e., there is no need of an infinite depth since $[\mathbf{T}_v(t_i)]_{i=1,...,j} = [\mathbf{T}_v^{2N_{t_i}-1}(t_i)]_{i=1,...,j}$. Considering the finite domain $\mathcal{G}_\mathcal{T}$ with $N = \max_{G_T^S \in \mathcal{G}_\mathcal{T}} |G_T^S|$, using a depth of $2N - 1$, we are sure that every unfolding three contains all the necessary information. Thus, from now on, we will assume that the depth of the unfolding trees is $2N - 1$. For sake of simplicity of notation, we will continue to use the notation $[\mathbf{T}_v(t_i)]_{i=1,...,j}$.

The main idea behind the proof is to design an STGNN that can encode the sequence of unfolding trees $[\mathbf{T}_v(t_i)]_{i=1,\dots,j}$ into the node attributes at t_j, i.e., $\mathbf{q}_v(t_j) = \#_{t_j}([\mathbf{T}_v(t_i)]_{i=1,\dots,j})$. To implement the encoding that could fit the definition of the STGNN, two coding functions are needed: the ∇ function, which takes as input an unfolding tree, and $\#_{t_j}$ which takes as input a sequence of unfolding trees up to time t_j (notice that the ∇ and $\#_{t_0}$ are the same function since at time t_0 the dynamic unfolding three is composed by just one unfolding three). The composition of these functions is used to define the node's attributes, and the STGNN can produce the desired output by using this encoded information as follows:

$$\begin{aligned}\mathbf{q}_v(0) &= \mathbf{x}_v^L(0) = \#_{t_0}\left(\nabla^{-1}(\mathbf{x}_v^L(0))\right)\\ \mathbf{q}_v(t_j) &= \#_{t_j}\left(\mathrm{APPEND}_{t_j}\left(\#_{t_{j-1}}^{-1}(\mathbf{q}_v(t_{j-1})), \nabla^{-1}(\mathbf{x}_v^L(t_j))\right)\right)\end{aligned} \tag{A.7}$$

where $\mathbf{x}_v^L(t_j)$ is the final representation produced by L layers of GNN.The auxiliar function APPEND_{t_j} and the ∇, $\#_{t_j}$ coding functions are defined in the following.

The $\mathbf{APPEND}_{t_j}$ function

Let $\mathcal{T}(v)$ be the domain of the unfolding trees with root v. The function $\mathrm{APPEND}_{t_j} : \{\mathbf{T}_v(t_i), i = 1, ..., j-1\} \cup \emptyset \times \mathcal{T}(v) \to \{\mathbf{T}_v(t_i), i = 1, ..., j\}$ is defined as follows:

$$\begin{aligned}&\mathrm{APPEND}_{t_0}\big(\emptyset, \mathbf{T}_v(0)\big) := \mathbf{T}_v(0)\\ &\mathrm{APPEND}_{t_j}\big(\big(\mathbf{T}_v(0), \dots, \mathbf{T}_v(t_j-1)\big), \mathbf{T}_v(t_j)\big)\\ &:= \big(\mathbf{T}_v(0), \dots, \mathbf{T}_v(t_j-1), \mathbf{T}_v(t_j)\big)\end{aligned}$$

Intuitively, this function appends the unfolding tree snapshot of the node v at time t_j to the sequence of the unfolding trees of that node at the previous timestamps.
In the following, the coding functions are defined; their existence and injectiveness are provided by construction.

The ∇ Coding Function

Let $\nabla := \mu_\nabla \circ \nu_\nabla$ be a composition of any two injective functions μ_∇ and ν_∇ with the following properties:

- μ_∇ is an injective function from the domain of static unfolding trees, calculated on the nodes in the STG, to the Cartesian product $\mathbb{N} \times \mathbb{N}^P \times \mathbb{Z}^A$, where P is the maximum number of nodes a tree could have.

  Intuitively, in the Cartesian product, $\mathbb{N}$ represents the tree structure, $\mathbb{N}^P$ denotes the node numbering, while, for each node, an integer vector in $\mathbb{Z}^A$ is used to encode the node attributes. Notice that μ_∇ exists and is injective since the maximal information contained in an unfolding tree is given by the union of all its node attributes and all its structural information, which just equals the dimension of the codomain of μ_∇.
- ν_∇ is an injective function from $\mathbb{N}^{P+1} \times \mathbb{Z}^A$ to $\mathbb{R}$, whose existence is guaranteed by the cardinality theory. Since μ_{∇_t} and ν_{∇_t} are injective, also the existence and the injectiveness of ∇_t is ensured.

The $\#_t$ Coding Functions

Similarly to ∇, the functions $\#_{t_j} := \mu_{\#_{t_j}} \circ \nu_{\#_{t_j}}$ are composed by two functions $\mu_{\#_{t_j}}$ and $\nu_{\#_{t_j}}$ with the following properties:

- $\mu_{\#_{t_j}}$ is an injective function from the domain of the dynamic unfolding trees to the Cartesian product $\mathbb{N}^{t_j} \times \mathbb{N}^{t_j P_{t_j}} \times \mathbb{Z}^{t_j A} = \mathbb{N}^{t_j (P_{t_j}+1)} \times \mathbb{Z}^{t_j A}$, where P_{t_j} is the maximum number of nodes a tree could have at time t_j.
- $\nu_{\#_{t_j}}$ is an injective function from $\mathbb{N}^{t_j (P+1)} \times \mathbb{Z}^{t_j A}$ to $\mathbb{R}$, whose existence is guaranteed by the cardinality theory. Since $\mu_{\#_{t_j}}$ and $\nu_{\#_{t_j}}$ are injective, also the existence and the injectiveness of $\#_{t_j}$ are ensured.

The REC, $\texttt{AGGREGATE}^{(l)}_{t_j}$, $\texttt{COMBINE}^{(l)}_{t_j}$ functions
Following the STGNN structure 3.2, the recursive function REC has to satisfy:

$$\texttt{REC}\big(\mathbf{q}_v(t_{j-1}), \mathbf{x}_v^L(t_j)\big) = \#_{t_j}([\mathbf{T}_v(t_i)]_{i=1,\dots,j}) = \mathbf{q}_v(t_j),$$

where the $\mathbf{x}_v^L(t_j)$ is the representation of node v at time t_j produced by the final layer of the GNN. Further, the functions $\texttt{AGGREGATE}^{(l)}_{t_j}$ and $\texttt{COMBINE}^{(l)}_{t_j}$ – following the proof in [217] – must satisfy

$$\begin{aligned}
&\nabla(\mathbf{T}_v^l(t_j)) = \mathbf{x}_v^l(t_j) = \\
&\texttt{COMBINE}^{(l)}_{t_j}\left(\mathbf{x}_v^{l-1}(t_j), \texttt{AGGREGATE}^{(l)}_{t_j}\left(\{\mathbf{x}_u^{l-1}(t_j)\}_{u\in\mathcal{N}_{t_j}[v]}\right)\right) \\
&= \texttt{COMBINE}^{(l)}_{t_j}\left(\nabla(\mathbf{T}_v^{l-1}(t_j)), \texttt{AGGREGATE}^{(l)}_{t_j}(\{\nabla(\mathbf{T}_u^{l-1}(t_j))\}_{u\in\mathcal{N}_{t_j}[v]})\right)
\end{aligned}$$

$\forall\, l \leq 2N-1$ and $\forall\, t_j$, where $\mathbf{x}_v^l(t_j)$ is the representation produced by the l-th intermediate layer of the GNN.

For example, the trees can be collected into the coding of a new tree, i.e.,

$$\texttt{AGGREGATE}^{(l)}_{t_j}(\nabla(\mathbf{T}_u^{l-1}(t_j)), u \in \mathcal{N}_{t_j}[v]) = \nabla(\cup_{u\in\mathcal{N}_{t_j}[v]}\nabla^{-1}(\nabla(\mathbf{T}_u^{l-1}(t_j)))),$$

where $\cup_{u\in\mathcal{N}_{t_j}[v]}$ denotes an operator that constructs a tree with a root having void attributes from a set of subtrees (see Fig. A.1). Then, $\texttt{COMBINE}^{(l)}_{t_j}$ assigns the correct attributes to the root by extracting them from $\mathbf{T}_v^{l-1}(t_j)$, i.e.,

$$\texttt{COMBINE}^{(l)}_{t_j}(\nabla(\mathbf{T}_v^{l-1}(t_j)), b) = \nabla(\texttt{ATTACH}(\nabla^{-1}(\nabla(\mathbf{T}_v^{l-1}(t_j))), \nabla^{-1}(b))),$$

where ATTACH is an operator that returns a tree constructed by replacing the attributes of the root in the latter tree with those of the former tree and b is the result of the $\texttt{AGGREGATE}^{(l)}_{t_j}$ function.

The READOUT function
Eventually, READOUT must satisfy:

$$\kappa(\cdot) := \texttt{READOUT}(\#_{t_j}(\cdot))$$

so that, ultimately,

$$\begin{aligned}
&\mathrm{dyn}(t_j, G_T^S, v) = \\
&\texttt{READOUT}\left(\#_{t_j}\left(\texttt{APPEND}_{t_j}\left(\#^{-1}_{t_{j-1}}(\mathbf{q}_v(t_{j-1})), \nabla^{-1}(\mathbf{x}_v^L(t_j))\right)\right)\right).
\end{aligned}$$

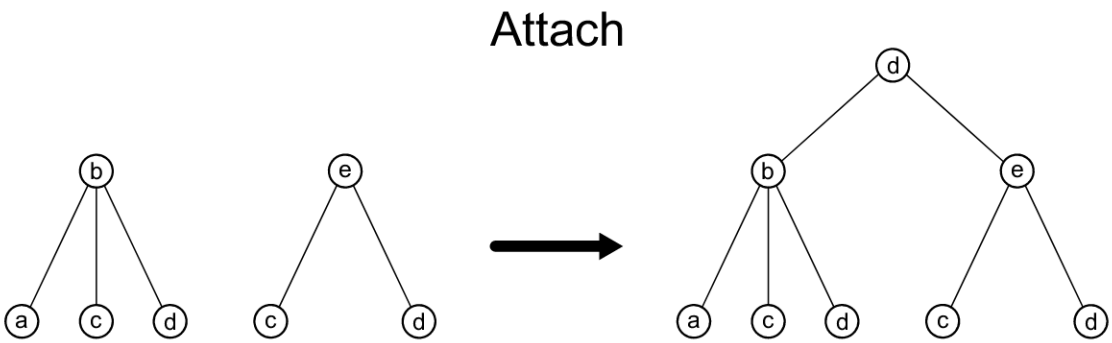


Figure A.1: The ATTACH operator on trees.

Notice that, with this definition, $\texttt{AGGREGATE}_{t_j}^{(l)}$, $\texttt{COMBINE}_{t_j}^{(l)}$, and $\texttt{READOUT}$ may not be differentiable. Nevertheless, Eq. (A.6) has to be satisfied only for a finite number of graphs. Thus, we can specify other functions $\overline{\texttt{AGGREGATE}_{t_j}}^{(l)}$, $\overline{\texttt{COMBINE}_{t_j}}^{(l)}$, and $\overline{\texttt{READOUT}}$, which produce exactly the same computations, but that can be extended to the rest of their domain so that they are continuously differentiable. Obviously, such an extension exists since those functions are only constrained to interpolate a finite number of points [1]

□

[1]Notice that a similar extension can also be applied to the coding function ∇ and to the decoding function ∇^{-1}. In this case, the coding function is not injective on the whole domain, but only on the graphs mentioned in the theorem.

Bibliography

[1] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[2] A. Mehrish, N. Majumder, R. Bharadwaj, R. Mihalcea, and S. Poria, "A review of deep learning techniques for speech processing," *Information Fusion*, p. 101869, 2023.

[3] R. Prabhavalkar, T. Hori, T. N. Sainath, R. Schlüter, and S. Watanabe, "End-to-end speech recognition: A survey," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

[4] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[5] J. Chai, H. Zeng, A. Li, and E. W. Ngai, "Deep learning in computer vision: A critical review of emerging techniques and application scenarios," *Machine Learning with Applications*, vol. 6, p. 100134, 2021.

[6] P. Andreini, G. Ciano, S. Bonechi, C. Graziani, V. Lachi, A. Mecocci, A. Sodi, F. Scarselli, and M. Bianchini, "A two-stage gan for high-resolution retinal image generation and segmentation," *Electronics*, vol. 11, no. 1, p. 60, 2021.

[7] N. Pancino, C. Graziani, V. Lachi, M. L. Sampoli, E. Ștefănescu, M. Bianchini, and G. M. Dimitri, "A mixed statistical and machine learning approach for the analysis of multimodal trail making test data," *Mathematics*, vol. 9, no. 24, p. 3159, 2021.

[8] M. Gao, F. Zheng, J. J. Yu, C. Shan, G. Ding, and J. Han, "Deep learning for video object segmentation: a review," *Artificial Intelligence Review*, vol. 56, no. 1, pp. 457–531, 2023.

[9] P. Veličković, "Everything is connected: Graph neural networks," *Current Opinion in Structural Biology*, vol. 79, p. 102538, 2023.

[10] B. Arregui-García, A. Longa, Q. F. Lotito, S. Meloni, and G. Cencetti, "Patterns in temporal networks with higher-order egocentric structures," *Entropy*, vol. 26, no. 3, p. 256, 2024.

[11] M. Tubaishat and S. Madria, "Sensor networks: an overview," *IEEE potentials*, vol. 22, no. 2, pp. 20–23, 2003.

[12] P. Badia-i Mompel, L. Wessels, S. Müller-Dott, R. Trimbour, R. O. Ramirez Flores, R. Argelaguet, and J. Saez-Rodriguez, "Gene regulatory network inference in the era of single-cell multi-omics," *Nature Reviews Genetics*, vol. 24, no. 11, pp. 739–754, 2023.

[13] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[14] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR 2016*, 2016.

[15] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[16] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48 550, 2017.

[17] 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.

[18] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He *et al.*, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Transactions on Recommender Systems*, vol. 1, no. 1, pp. 1–51, 2023.

[19] H. P. Samoaa, A. Longa, M. Mohamad, M. H. Chehreghani, and P. Leitner, "Tep-gnn: Accurate execution time prediction of functional tests using graph neural networks," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2022, pp. 464–479.

[20] P. Samoaa, L. Aronsson, A. Longa, P. Leitner, and M. H. Chehreghani, "A unified active learning framework for annotating graph data with application to software source code performance prediction," *arXiv preprint arXiv:2304.13032*, 2023.

[21] G. Giacomini, C. Graziani, V. Lachi, P. Bongini, N. Pancino, M. Bianchini, D. Chiarugi, A. Valleriani, and P. Andreini, "A neural network approach for the analysis of reproducible ribo–seq profiles," *Algorithms*, vol. 15, no. 8, p. 274, 2022.

[22] P. Bongini, N. Pancino, V. Lachi, C. Graziani, G. Giacomini, P. Andreini, and M. Bianchini, "Point-wise ribosome translation speed prediction with recurrent neural networks," *Mathematics*, vol. 12, no. 3, p. 465, 2024.

[23] P. Bongini, M. Bianchini, and F. Scarselli, "Molecular generative graph neural networks for drug discovery," *Neurocomputing*, vol. 450, pp. 242–252, 2021.

[24] P. Bongini, E. Messori, N. Pancino, and M. Bianchini, "A deep learning approach to the prediction of drug side–effects on molecular graphs," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2023.

[25] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *International Conference on Learning Representations*, 2020.

[26] G. Ciano, A. Rossi, M. Bianchini, and F. Scarselli, "On inductive–transductive learning with graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 2, pp. 758–769, 2021.

[27] A. Rossi, M. Tiezzi, G. M. Dimitri, M. Bianchini, M. Maggini, and F. Scarselli, "Inductive–transductive learning with graph neural networks," in *Artificial Neural Networks in Pattern Recognition: 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19–21, 2018, Proceedings 8*. Springer, 2018, pp. 201–212.

[28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[29] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," *Advances in neural information processing systems*, vol. 32, 2019.

[30] F. Scarselli, A. C. Tsoi, and M. Hagenbuchner, "The vapnik–chervonenkis dimension of graph and recursive neural networks," *Neural Networks*, vol. 108, pp. 248–259, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608018302363

[31] C. Morris, F. Geerts, J. Tönshoff, and M. Grohe, "Wl meet vc," in *International Conference on Machine Learning*. PMLR, 2023, pp. 25 275–25 302.

[32] C. Gao, X. Wang, X. He, and Y. Li, "Graph neural networks for recommender system," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1623–1625.

[33] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM CSUR*, 2022.

[34] S. Deng, H. Rangwala, and Y. Ning, "Learning dynamic context graphs for predicting social events," in *ACM SIGKDD*, 2019.

[35] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 417–426.

[36] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, vol. 207, p. 117921, 2022.

[37] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

[38] M. Cardia, M. Luca, and L. Pappalardo, "Enhancing crowd flow prediction in various spatial and temporal granularities," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 1251–1259.

[39] A. Longa, G. Cencetti, B. Lepri, and A. Passerini, "An efficient procedure for mining egocentric temporal motifs," *Data Mining and Knowledge Discovery*, 2022.

[40] G. Mauro, M. Luca, A. Longa, B. Lepri, and L. Pappalardo, "Generating mobility networks with generative adversarial networks," *EPJ Data Science*, vol. 11, no. 1, p. 58, 2022.

[41] S. Gao, "Spatio-temporal analytics for exploring human mobility patterns and urban dynamics in the mobile age," *Spatial Cognition & Computation*, vol. 15, no. 2, pp. 86–114, 2015.

[42] M. Luca, G. Barlacchi, B. Lepri, and L. Pappalardo, "A survey on deep learning for human mobility," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–44, 2021.

[43] G. Cencetti, G. Santin, A. Longa, E. Pigani, A. Barrat, C. Cattuto, S. Lehmann, M. Salathe, and B. Lepri, "Digital proximity tracing on empirical contact networks for pandemic control," *Nature Communications*, 2021.

[44] M. K. So, A. Tiwari, A. M. Chu, J. T. Tsang, and J. N. Chan, "Visualizing covid-19 pandemic risk through network connectedness," *International Journal of Infectious Diseases*, vol. 96, pp. 558–561, Jul 2020.

[45] N. Ahmed, L. Chen, Y. Wang, B. Li, Y. Li, and W. Li, "DeepEye: Link prediction in dynamic networks based on non-negative matrix factorization," *Big Data Mining and Analytics*, 2018.

[46] A. Longa, G. Cencetti, S. Lehmann, A. Passerini, and B. Lepri, "Generating fine-grained surrogate temporal networks," *Communications Physics*, vol. 7, no. 1, p. 22, 2024.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[48] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1416–1424.

[49] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[50] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5115–5124.

[51] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[52] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *NeurIPS*, 2018.

[53] L. Cai and S. Ji, "A multi-scale approach for graph link prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 3308–3315.

[54] M. Zhang and Y. Chen, "Weisfeiler-lehman neural machine for link prediction," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 575–583.

[55] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," *arXiv preprint arXiv:2002.07962*, 2020.

[56] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," *NeurIPS*, vol. 32, 2019.

[57] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*, 2020.

[58] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[59] X. Wang, D. Lyu, M. Li, Y. Xia, Q. Yang, X. Wang, X. Wang, P. Cui, Y. Yang, B. Sun *et al.*, "APAN: Asynchronous propagation attention network for real-time temporal graph embedding," in *SIGMOD*, 2021.

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

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[62] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[63] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *International conference on machine learning*. PMLR, 2020, pp. 874–883.

[64] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *Advances in neural information processing systems*, vol. 31, 2018.

[65] F. M. Bianchi and V. Lachi, "The expressive power of pooling in graph neural networks," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[66] S. Beddar-Wiesing, G. A. D'Inverno, C. Graziani, V. Lachi, A. Moallemy-Oureh, F. Scarselli, and J. M. Thomas, "Weisfeiler–lehman goes dynamic: An analysis of the expressive power of graph neural networks for attributed and dynamic graphs," *Neural Networks*, p. 106213, 2024.

[67] V. Lachi, A. Moallemy-Oureh, A. Roth, and P. Welke, "Graph pooling provably improves expressivity," in *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

[68] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini, "Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities," *arXiv preprint arXiv:2302.01018*, 2023.

[69] V. Lachi, F. Ferrini, A. Longa, B. Lepri, and A. Passerini, "A simple and expressive graph neural network based method for structural link representation," in *ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling*, 2024.

[70] S. Beddar-Wiesing, G. A. D'Inverno, C. Graziani, V. Lachi, A. Moallemy-Oureh *et al.*, "On the extension of the weisfeiler-lehman hierarchy by wl tests for arbitrary graphs," in *18th International Workshop on Mining and Learning with Graphs*, 2022.

[71] P. Andreini, S. Bonechi, G. Ciano, C. Graziani, V. Lachi, N. Nikoloulopoulou, M. Bianchini, and F. Scarselli, "Multi-stage synthetic image generation for the semantic segmentation of medical images," in *Artificial Intelligence and Machine Learning for Healthcare: Vol. 1: Image and Data Analytics*. Springer, 2022, pp. 79–104.

[72] F. Scarselli, S. L. Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking web pages," in *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*. IEEE, 2005, pp. 666–672.

[73] A. Leman and B. Weisfeiler, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsiya*, vol. 2, no. 9, pp. 12–16, 1968.

[74] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[75] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=dFbKQaRk15w

[76] C. Graziani, T. Drucks, M. Bianchini, F. Scarselli, T. Gärtner *et al.*, "No pain no gain: More expressive gnns with paths," in *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

[77] R. Abboud, İ. İ. Ceylan, M. Grohe, and T. Lukasiewicz, "The surprising power of graph neural networks with random node initialization," in *Proceedings of the Thirtieth International Joint Conference on Artifical Intelligence (IJCAI)*, 2021.

[78] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021.

[79] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.

[80] S. Kumar, X. Zhang, and J. Leskovec, "Learning dynamic embeddings from temporal interactions," *arXiv preprint arXiv:1812.02289*, 2018.

[81] S. S. Dasgupta, S. N. Ray, and P. Talukdar, "Hyte: Hyperplane-based temporally aware knowledge graph embedding," in *Proceedings of the 2018 conference on empirical methods in natural language processing*, 2018, pp. 2001–2011.

[82] A. Taheri, K. Gimpel, and T. Berger-Wolf, "Learning to represent the evolution of dynamic graphs with recurrent models," in *Companion proceedings of the 2019 world wide web conference*, 2019, pp. 301–307.

[83] S. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey." *Journal of Machine Learning Research*, 2020.

[84] C. Barros, M. Mendonça, A. Vieira, and A. Ziviani, "A survey on embedding dynamic graphs," *ACM CSUR*, 2021.

[85] Y. Luo and P. Li, "Neighborhood-aware scalable temporal network representation learning," *arXiv preprint arXiv:2209.01084*, 2022.

[86] S. Gupta and S. Bedathur, "A survey on temporal graph representation learning and generative modeling," *arXiv preprint arXiv:2208.12126*, 2022.

[87] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," in *AAAI*, 2020.

[88] H. Zhou, D. Zheng, I. Nisa, V. Ioannidis, X. Song, and G. Karypis, "TGL: A general framework for temporal GNN training on billion-scale graphs," *arXiv preprint arXiv:2203.14883*, 2022.

[89] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *ACM SIGIR*, 2020.

[90] A. Micheli and D. Tortorella, "Discrete-time dynamic graph echo state networks," *Neurocomputing*, vol. 496, pp. 85–95, 2022.

[91] M. V. D. Heuvel, R. Mandl, C. Stam., R. Kahn, P. Hulshoff, and E. Hilleke, "Aberrant frontal and temporal complex network structure in schizophrenia: A graph theoretical analysis," *Journal of Neuroscience*, 2010.

[92] R. Gao, J. Yan, P. Li, and L. Chen, "Detecting the critical states during disease development based on temporal network flow entropy," *Briefings in Bioinformatics*, 2022.

[93] R. Keisler, "Forecasting global weather with graph neural networks," *arXiv preprint arXiv:2202.07575*, 2022.

[94] I. McBrearty and G. Beroza, "Earthquake location and magnitude estimation with graph neural networks," in *IEEE ICIP*, 2022.

[95] A. Cini, I. Marisca, F. Bianchi, and C. Alippi, "Scalable spatiotemporal graph neural networks," *arXiv preprint arXiv:2209.06520*, 2022.

[96] M. Qin and D. Yeung, "Temporal link prediction: A unified framework, taxonomy, and review," *arXiv preprint arXiv:2210.08765*, 2022.

[97] J. You, T. Du, and J. Leskovec, "ROLAND: graph learning framework for dynamic graphs," in *ACM SIGKDD*, 2022.

[98] X. Ru, J. M. Moore, X.-Y. Zhang, Y. Zeng, and G. Yan, "Inferring patient zero on temporal networks via graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 8, 2023, pp. 9632–9640.

[99] P. Hiram Guzzi, F. Petrizzelli, and T. Mazza, "Disease spreading modeling and analysis: A survey," *Briefings in Bioinformatics*, vol. 23, no. 4, p. bbac230, 2022.

[100] A. Koher, H. H. Lentz, J. P. Gleeson, and P. Hövel, "Contact-based model for epidemic spreading on temporal networks," *Physical Review X*, vol. 9, no. 3, p. 031017, 2019.

[101] A. Darbon, D. Colombi, E. Valdano, L. Savini, A. Giovannini, and V. Colizza, "Disease persistence on temporal contact networks accounting for heterogeneous infectious periods," *Royal Society open science*, vol. 6, no. 1, p. 181404, 2019.

[102] A. Darbon, E. Valdano, C. Poletto, A. Giovannini, L. Savini, L. Candeloro, and V. Colizza, "Network-based assessment of the vulnerability of italian regions to bovine brucellosis," *Preventive veterinary medicine*, vol. 158, pp. 25–34, 2018.

[103] K. Ljubičić, A. Merćep, and Z. Kostanjčar, "Analysis of complex customer networks: A real-world banking example," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE, 2022, pp. 321–326.

[104] A. Rosyidah, I. Surjandari *et al.*, "Exploring customer data using spatio-temporal analysis: Case study of fixed broadband provider," *International Journal of Applied Science and Engineering*, vol. 16, no. 2, pp. 133–147, 2019.

[105] G. Jacquez, J. Meliker, R. Rommel, and P. Goovaerts, "Exposure reconstruction using space-time information technology," *Encyclopedia of Environmental Health*, pp. 793–804, 2019.

[106] J. R. Meliker, G. M. Jacquez, P. Goovaerts, G. Copeland, and M. Yassine, "Spatial cluster analysis of early stage breast cancer: a method for public health practice using cancer registry data," *Cancer Causes & Control*, vol. 20, pp. 1061–1069, 2009.

[107] D. C. Wheeler, "A comparison of spatial clustering and cluster detection techniques for childhood leukemia incidence in ohio, 1996–2003," *International journal of health geographics*, vol. 6, no. 1, pp. 1–16, 2007.

[108] A. Ozonoff, T. Webster, V. Vieira, J. Weinberg, D. Ozonoff, and A. Aschengrau, "Cluster detection methods applied to the upper cape cod cancer data," *Environmental Health*, vol. 4, pp. 1–9, 2005.

[109] J. Enright and R. Rowland, "Epidemics on dynamic networks," *Epidemics*, 2018.

[110] A. Myers, D. Muñoz, F. A. Khasawneh, and E. Munch, "Temporal network analysis using zigzag persistence," *EPJ Data Science*, vol. 12, no. 1, p. 6, 2023.

[111] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[112] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *International conference on information processing in medical imaging*. Springer, 2017, pp. 146–157.

[113] J. Kim and C. D. Scott, "Robust kernel density estimation," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2529–2565, 2012.

[114] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova, "Anomaly detection in dynamic networks: a survey," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 7, no. 3, pp. 223–247, 2015.

[115] L. Akoglu and C. Faloutsos, "Event detection in time series of mobile communication graphs," in *Army science conference*, vol. 1, 2010, p. 141.

[116] W. Khan and M. Haroon, "A pilot study and survey on methods for anomaly detection in online social networks," in *Human-Centric Smart Computing: Proceedings of ICHCSC 2022*. Springer, 2022, pp. 119–128.

[117] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han, "On community outliers and their efficient detection in information networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 813–822.

[118] T. Ji, D. Yang, and J. Gao, "Incremental local evolutionary outlier detection for dynamic social networks," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II 13*. Springer, 2013, pp. 1–15.

[119] H. Zhang, Y. Zheng, and Y. Yu, "Detecting urban anomalies using multiple spatio-temporal data sources," *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, vol. 2, no. 1, pp. 1–18, 2018.

[120] L. Deng, D. Lian, Z. Huang, and E. Chen, "Graph convolutional adversarial networks for spatiotemporal anomaly detection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 6, pp. 2416–2428, 2022.

[121] D. Savage, X. Zhang, X. Yu, P. Chou, and Q. Wang, "Anomaly detection in online social networks," *Social networks*, vol. 39, pp. 62–70, 2014.

[122] N. A. Heard, D. J. Weston, K. Platanioti, and D. J. Hand, "Bayesian anomaly detection methods for social networks," *The Annals of Applied Statistics*, vol. 4, no. 2, pp. 645 – 662, 2010.

[123] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[124] M. Mongiovi, P. Bogdanov, R. Ranca, E. E. Papalexakis, C. Faloutsos, and A. K. Singh, "Netspot: Spotting significant anomalous regions on dynamic networks," in *Proceedings of the 2013 Siam international conference on data mining*. SIAM, 2013, pp. 28–36.

[125] D. Zambon, C. Alippi, and L. Livi, "Concept drift and anomaly detection in graph streams," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5592–5605, 2018.

[126] T. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[127] M. Yin and M. Zhou, "Semi-implicit variational inference," in *ICML*, 2018.

[128] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[129] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NeurIPS*, 2008.

[130] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. López, N. Collignon *et al.*, "Pytorch Geometric Temporal: Spatiotemporal signal processing with neural machine learning models," in *ACM CIKM*, 2021.

[131] M. Guan, A. Iyer, and T. Kim, "DynaGraph: dynamic graph neural networks at scale," in *ACM SIGMOD22 GRADES-NDA*, 2022.

[132] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open Graph Benchmark: Datasets for machine learning on graphs," *NeurIPS*, vol. 33, pp. 22 118–22 133, 2020.

[133] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," *Advances in neural information processing systems*, vol. 33, pp. 19 620–19 631, 2020.

[134] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[135] A. Longa, S. Azzolin, G. Santin, G. Cencetti, P. Liò, B. Lepri, and A. Passerini, "Explaining the explainers in graph neural networks: a comparative study," *arXiv preprint arXiv:2210.15304*, 2022.

[136] S. Azzolin, A. Longa, P. Barbiero, P. Lio, and A. Passerini, "Global explainability of gnns via logic combination of learned concepts," in *The Eleventh International Conference on Learning Representations*.

[137] W. Xia, M. Lai, C. Shan, Y. Zhang, X. Dai, X. Li, and D. Li, "Explaining temporal graph models through an explorer-navigator framework," in *The Eleventh International Conference on Learning Representations*, 2022.

[138] M. N. Vu and M. T. Thai, "On the limit of explaining black-box temporal graph neural networks," *arXiv preprint arXiv:2212.00952*, 2022.

[139] W. He, M. N. Vu, Z. Jiang, and M. T. Thai, "An explainer for temporal graph neural networks," in *GLOBECOM - IEEE Global Communications Conference 2022*. IEEE, 2022, pp. 6384–6389.

[140] C. Bodnar, F. D. Giovanni, B. Chamberlain, P. Liò, and M. Bronstein, "Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs," in *ICLR*, 2022.

[141] J. Topping, F. D. Giovanni, B. Chamberlain, X. Dong, and M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *arXiv preprint arXiv:2111.14522*, 2021.

[142] O. Zaghen, A. Longa, S. Azzolin, L. Telyatnikov, P. Lio, and A. Passerini, "Sheaf diffusion goes nonlinear: Enhancing gnns with adaptive sheaf laplacians," in *ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling*, 2024.

[143] M. Yang, Z. Meng, and I. King, "Featurenorm: L2 feature normalization for dynamic graph embedding," in *ICDM*, 2020.

[144] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating scientific knowledge with machine learning for engineering and environmental systems," *ACM Computing Surveys*, vol. 55, no. 4, 2022.

[145] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[146] S. Cuomo, V. D. Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics–informed neural networks: where we are and what's next," *Journal of Scientific Computing*, 2022.

[147] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia, "Learning mesh-based simulation with graph networks," in *ICLR*, 2021.

[148] H. Gao, M. Zahr, and J. Wang, "Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering*, 2022.

[149] J. H. Faghmous and V. Kumar, "A Big Data Guide to Understanding Climate Change: The Case for Theory-Guided Data Science," *Big Data*, vol. 2, no. 3, 2014.

[150] F. Ferrini, A. Longa, A. Passerini, and M. Jaeger, "Meta-path learning for multi-relational graph neural networks," in *Learning on Graphs Conference*. PMLR, 2024, pp. 2–1.

[151] M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You, and J. Leskovec, "Relational deep learning: Graph representation learning on relational databases," *arXiv preprint arXiv:2312.04615*, 2023.

[152] R. Sato, "A survey on the expressive power of graph neural networks," *arXiv preprint arXiv:2003.04078*, 2020.

[153] J. Gao and B. Ribeiro, "On the equivalence between temporal and static equivariant graph representations," in *International Conference on Machine Learning*. PMLR, 2022, pp. 7052–7076.

[154] G. D'Inverno, M. Bianchini, M. Sampoli, and F. Scarselli, "A new perspective on the approximation capability of GNNs," *arXiv preprint arXiv:2106.08992*, 2021.

[155] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[156] B. Hammer, "On the approximation capability of recurrent neural networks," *Neurocomputing*, vol. 31, no. 1-4, pp. 107–123, 2000.

[157] W. Azizian and M. Lelarge, "Expressive power of invariant and equivariant graph neural networks," *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. [Online]. Available: https://openreview.net/forum?id=lxHgXYN4bwl

[158] S. Kiefer and B. D. McKay, "The iteration number of colour refinement," *arXiv preprint arXiv:2005.10182*, 2020.

[159] A. Krebs and O. Verbitsky, "Universal covers, color refinement, and two-variable counting logic: Lower bounds for the depth," in *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 2015, pp. 689–700.

[160] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[161] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *ICLR*, 2018.

[162] Y. Wang, Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," *arXiv preprint arXiv:2101.05974*, 2021.

[163] Z. Liu, D. Zhou, Y. Zhu, J. Gu, and J. He, "Towards fine-grained temporal network representation via time-reinforced random walk," in *AAAI*, vol. 34, 2020, pp. 4973–4980.

[164] M. Jin, Y.-F. Li, and S. Pan, "Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs," *Advances in Neural Information Processing Systems*, vol. 35, pp. 19 874–19 886, 2022.

[165] K. Sato, M. Oka, A. Barrat, and C. Cattuto, "Dyane: dynamics-aware node embedding for temporal networks," *arXiv preprint arXiv:1909.05976*, 2019.

[166] P. Lewis, "Multivariate point processes," in *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1. University of California Press, 1972, p. 401.

[167] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *International Conference on Learning Representations*, 2019.

[168] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs," in *international conference on machine learning*. PMLR, 2017, pp. 3462–3471.

[169] Z. Daniele, C. Alippi *et al.*, "Az-whiteness test: a test for uncorrelated noise on spatio-temporal graphs," in *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022, pp. 1–17.

[170] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.

[171] D. Xu, J. Liang, W. Cheng, H. Wei, H. Chen, and X. Zhang, "Transformer-style relational reasoning with dynamic memory updating for temporal network modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4546–4554.

[172] D. Xu, W. Cheng, D. Luo, X. Liu, and X. Zhang, "Spatio-temporal attentive rnn for node classification in temporal attributed graphs." in *IJCAI*, 2019, pp. 3947–3953.

[173] F. L. Opolka, A. Solomon, C. Cangea, P. Veličković, P. Liò, and R. Hjelm, "Spatio-temporal deep graph infomax," *arXiv preprint arXiv:1904.06316*, 2019.

[174] B. Zhou, X. Liu, Y. Liu, Y. Huang, P. Lio, and Y. G. Wang, "Well-conditioned spectral transforms for dynamic graph representation," in *Learning on Graphs Conference*. PMLR, 2022, pp. 12–1.

[175] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Node classification in temporal graphs through stochastic sparsification and temporal structural convolution," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*. Springer, 2021, pp. 330–346.

[176] I. Marisca, A. Cini, and C. Alippi, "Learning to reconstruct missing data from spatiotemporal graphs with sparse observations," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32 069–32 082, 2022.

[177] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[178] J. Baek, M. Kang, and S. J. Hwang, "Accurate learning of graph representations with graph multiset pooling," in *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[179] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Hierarchical representation learning in graph neural networks with node decimation pooling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2195–2207, 2020.

[180] D. Bacciu, A. Conte, and F. Landolfi, "Graph pooling with maximum-weight k-independent sets," in *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 2023.

[181] H. Gao and S. Ji, "Graph u-nets," in *international conference on machine learning*. PMLR, 2019, pp. 2083–2092.

[182] Z. Ma, J. Xuan, Y. G. Wang, M. Li, and P. Liò, "Path integral based convolution and pooling for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 421–16 433, 2020.

[183] B. Knyazev, G. W. Taylor, and M. Amer, "Understanding attention and generalization in graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[184] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 2020.

[185] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[186] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *J. Mach. Learn. Res.*, vol. 24, pp. 127:1–127:21, 2023.

[187] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International conference on machine learning*. PMLR, 2019, pp. 3734–3743.

[188] E. Ranjan, S. Sanyal, and P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5470–5477.

[189] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in neural information processing systems*, vol. 30, 2017.

[190] D. Bacciu and L. Di Sotto, "A non-negative factorization approach to node pooling in graph convolutional neural networks," in *AI* IA 2019–Advances in Artificial Intelligence: XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19–22, 2019, Proceedings 18*. Springer, 2019, pp. 294–306.

[191] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, 2016.

[192] S. T. Barnard and H. D. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and experience*, vol. 6, no. 2, pp. 101–117, 1994.

[193] D. Mesquita, A. Souza, and S. Kaski, "Rethinking pooling in graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2220–2231, 2020.

[194] M. Guerra, I. Spinelli, S. Scardapane, and F. M. Bianchi, "Explainability in subgraphs-enhanced graph neural networks," *arXiv preprint arXiv:2209.07926*, 2022.

[195] F. Diehl, "Edge contraction pooling for graph neural networks," *arXiv preprint arXiv:1905.10990*, 2019.

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

[197] 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.

[198] F. M. Bianchi, C. Gallicchio, and A. Micheli, "Pyramidal reservoir graph neural network," vol. 470. Elsevier, 2022, pp. 389–404.

[199] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *AAAI Conference on Artificial Intelligence*, 2019.

[200] E. Luzhnica, B. Day, and P. Lio, "Clique pooling for graph classification," in *Representation Learning on Graphs and Manifolds Workshop*, 2019.

[201] C. Sanders, A. Roth, and T. Liebig, "Curvature-based pooling within graph neural networks," in *Mining and Learning with Graphs Workshop*, 2023.

[202] M. Fey, J.-G. Yuen, and F. Weichert, "Hierarchical inter-message passing for learning on molecular graphs," in *Graph Representation Learning and Beyond Workshop*, 2020.

[203] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[204] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 127, pp. 1–21, 2023.

[205] Y. Wang and M. Zhang, "Towards better evaluation of GNN expressiveness with BREC dataset," in *Submitted to The Twelfth International Conference on Learning Representations*, 2023, under review. [Online]. Available: https://openreview.net/forum?id=aqqE1yS3RY

[206] D. Neuen and P. Schweitzer, "Benchmark graphs for practical graph isomorphism," in *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, ser. LIPIcs, K. Pruhs and C. Sohler, Eds., vol. 87. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 60:1–60:14. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2017.60

[207] J.-Y. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification," *Combinatorica*, vol. 12, no. 4, pp. 389–410, 1992.

[208] Y. Gurevich and S. Shelah, "On finite rigid structures," *J. Symb. Log.*, vol. 61, no. 2, pp. 549–562, 1996. [Online]. Available: https://doi.org/10.2307/2275675

[209] A. Cini, I. Marisca, D. Zambon, and C. Alippi, "Graph deep learning for time series forecasting," *arXiv preprint arXiv:2310.15978*, 2023.

[210] J. Skarding, B. Gabrys, and K. Musial, "Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey," *IEEE Access*, vol. 9, pp. 79 143–79 168, 2021.

[211] C. Cai and Y. Wang, "A note on over-smoothing for graph neural networks," *arXiv preprint arXiv:2006.13318*, 2020.

[212] M. Black, Z. Wan, A. Nayyeri, and Y. Wang, "Understanding oversquashing in gnns through the lens of effective resistance," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2528–2547.

[213] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Revisiting graph neural networks for link prediction," 2021. [Online]. Available: https://openreview.net/forum?id=8q_ca26L1fz

[214] B. Srinivasan and B. Ribeiro, "On the equivalence between positional node embeddings and structural graph representations," *arXiv preprint arXiv:1910.00452*, 2019.

[215] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational Capabilities of Graph Neural Networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2008.

[216] J. M. Thomas, S. Beddar-Wiesing, and A. Moallemy-Oureh, "Graph Type Expressivity and Transformations," *arXiv:2109.10708*, 09 2021.

[217] D'Inverno, Giuseppe A. and Bianchini, Monica and Sampoli, Maria L. and Scarselli, Franco, "On the approximation capability of GNNs in node classification/regression tasks," *arXiv preprint arXiv:2106.08992*, 2021.