Hierarchical Representation Learning in Graph Neural Networks With Node Decimation Pooling

Filippo Maria Bianchi, Daniele Grattarola, *Student Member, IEEE*, Lorenzo Livi, *Member, IEEE*, and Cesare Alippi, *Fellow, IEEE*

***Abstract*— In graph neural networks (GNNs), pooling operators compute local summaries of input graphs to capture their global properties, and they are fundamental for building deep GNNs that learn hierarchical representations. In this work, we propose the Node Decimation Pooling (NDP), a pooling operator for GNNs that generates coarser graphs while preserving the overall graph topology. During training, the GNN learns new node representations and fits them to a pyramid of coarsened graphs, which is computed offline in a preprocessing stage. NDP consists of three steps. First, a node decimation procedure selects the nodes belonging to one side of the partition identified by a spectral algorithm that approximates the `MAXCUT` solution. Afterward, the selected nodes are connected with Kron reduction to form the coarsened graph. Finally, since the resulting graph is very dense, we apply a sparsification procedure that prunes the adjacency matrix of the coarsened graph to reduce the computational cost in the GNN. Notably, we show that it is possible to remove many edges without significantly altering the graph structure. Experimental results show that NDP is more efficient compared to state-of-the-art graph pooling operators while reaching, at the same time, competitive performance on a significant variety of graph classification tasks.**

***Index Terms*— Graph classification, graph neural networks (GNNs), graph pooling, Kron reduction, Maxcut optimization.**

I. Introduction

GENERATING hierarchical representations across the layers of a neural network is key to deep learning methods. This hierarchical representation is usually achieved through pooling operations, which progressively reduce the dimensionality of the inputs encouraging the network to learn high-level data descriptors. Graph Neural Networks (GNNs) are machine learning models that learn abstract representations of graph-structured data to solve a large variety of inference tasks [1]–[5]. Different from neural networks that process vectors, images, or sequences, the graphs processed by GNNs have arbitrary topology. As a consequence, standard pooling operations that leverage on the regular structure of the data and physical locality principles cannot be immediately applied to GNNs.

Graph pooling aggregates vertex features while reducing, at the same time, the underlying structure in order to maintain meaningful connectivity in the coarsened graph. By alternating graph pooling and message-passing (MP) operations [6], a GNN can gradually distill global properties from the graph, which are then used in tasks such as graph classification.

In this article, we propose *Node Decimation Pooling* (NDP), a pooling operator for GNNs. NDP is based on node decimation, a procedure developed in the field of graph signal processing for the design of multiscale graph filters [7]. In particular, we build upon the *multiresolution* framework [8] that consists of removing some nodes from a graph and then building a coarsened graph from the remaining ones. The NDP procedure that we propose precomputes off-line (i.e., before training) a *pyramid* of coarsened graphs, which are then used as support for the node representations computed at different levels of the GNN architecture.

The contributions of our work are as follows.

1) We introduce the NDP operator that allows to implement deep GNNs that have low complexity (in terms of execution time and memory requirements) and achieve high accuracy on several downstream tasks.
2) We propose a simple and efficient spectral algorithm that partitions the graph nodes in two sets by maximizing a `MAXCUT` objective. Such a partition is exploited to select the nodes to be discarded when coarsening the graph.
3) We propose a graph sparsification procedure that reduces the computational cost of MP operations applied after pooling and has a small impact on the representations learned by the GNN. In particular, we show both analytically and empirically that many edges can be removed without significantly altering the graph structure.

Manuscript received October 24, 2019; revised May 6, 2020 and September 18, 2020; accepted December 8, 2020. The work of Daniele Grattarola was supported by the Swiss National Science Foundation under Project 200021 172671. *(Corresponding author: Filippo Maria Bianchi.)*

Filippo Maria Bianchi is with the Department of Mathematics and Statistics, UiT The Arctic University of Norway, 9019 Tromsø, Norway, and also with the Norwegian Research Center (NORCE), 5008 Bergen, Norway (e-mail: filippo.m.bianchi@uit.no).

Daniele Grattarola is with the Faculty of Informatics, Università della Svizzera italiana, 6904 Lugano, Switzerland.

Lorenzo Livi is with the Department of Computer Science and Mathematics, University of Manitoba, Winnipeg, MB R3T 2N2, Canada, and also with the Department of Computer Science, University of Exeter, Exeter EX4 4QJ, U.K.

Cesare Alippi is with the Faculty of Informatics, Università della Svizzera italiana, 6900 Lugano, Switzerland, and also with the Department of Electronics, Information, and Bioengineering, Politecnico di Milan, 20133 Milan, Italy.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2020.3044146.

Digital Object Identifier 10.1109/TNNLS.2020.3044146

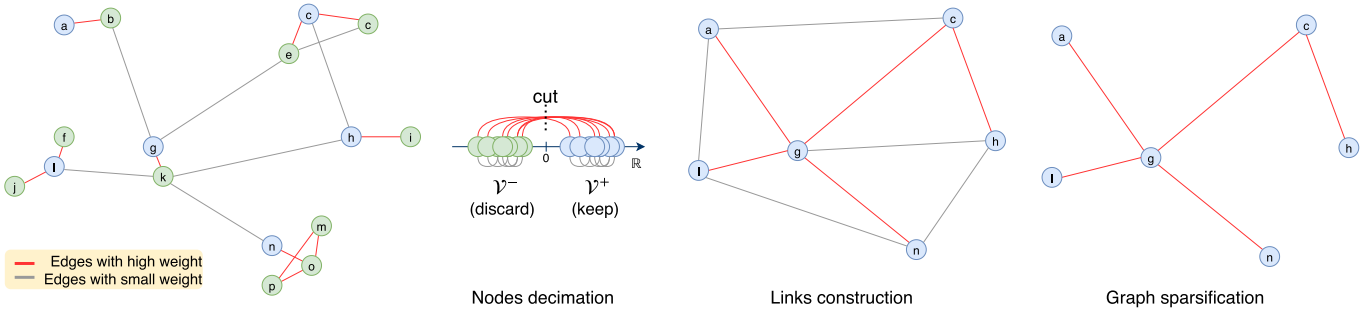

Fig. 1. Depiction of the proposed graph coarsening procedure. First, the nodes are partitioned in two sets according to a MAXCUT objective and then are decimated by dropping one of the two sets ($\mathcal{V}^-$). Then, a coarsened Laplacian is built by connecting the remaining nodes with a graph reduction procedure. Finally, the edges with low weights in the new adjacency matrix obtained from the coarsened Laplacian are dropped to make the resulting graph sparser.

When compared to other methods for graph pooling, NDP performs significantly better than other techniques that precompute the topology of the coarsened graphs, while it achieves a comparable performance with respect to state-of-the-art *feature-based* pooling methods. The latter, learn both the topology and the features of the coarsened graphs end-to-end via gradient descent, at the cost of larger model complexity and higher training time. The efficiency of NDP brings a significant advantage when GNNs are deployed in real-world scenarios subject to computational constraints, like in embedded devices and sensor networks.

This article is organized as follows: in Section II, we formalize the problem and introduce the nomenclature; in Section III, we present the proposed method; Section IV provides formal analyses and implementation details; related works are discussed in Section V, and Section VI reports experimental results. Further results and analyses are deferred to the supplementary material.

II. Preliminaries

Let $G = \{\mathcal{V}, \mathcal{E}\}$ be a graph with node set $\mathcal{V}$, $|\mathcal{V}| = N$, and edge set $\mathcal{E}$ described by a symmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$. Define as *graph signal* $\mathbf{X} \in \mathbb{R}^{N\times F}$ the matrix containing the features of the nodes in the graph (the ith row of $\mathbf{X}$ corresponds with the features $\mathbf{x}_i \in \mathbb{R}^F$ of the ith node). For simplicity, we will only consider undirected graphs without edge annotations.

Let $\mathbf{L} = \mathbf{D} - \mathbf{A}$ be the Laplacian of the graph, where $\mathbf{D}$ is a diagonal degree matrix s.t. d_{ii} is the degree of node i. We also define the symmetric Laplacian as $\mathbf{L}_s = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The Laplacian characterizes the dynamics of a diffusion process on the graph and plays a fundamental role in the proposed graph reduction procedure. We note that in the presence of directed edges it is still possible to obtain a symmetric and positive-semidefinite Laplacian [9], [10] for which the derivations presented in this article hold.

We consider a GNN composed of a stack of MP layers, each one followed by a graph pooling operation. The (l)th pooling operation reduces N_l nodes to $N_{l+i} < N_l$, producing a pooled version of the node features $\mathbf{X}^{(l+1)} \in \mathbb{R}^{N_{l+1}\times F_{l+1}}$ and adjacency matrix $\mathbf{A}^{(l+1)} \in \mathbb{R}^{N_{l+1}\times N_{l+1}}$. To implement the MP layer, we consider a simple formulation that operates on the first-order neighborhood of each node and accounts for the original node features through a separate set of weights acting as a layer-wise skip connection. The computation carried out by the (j)th MP layer is given by

$$\begin{aligned}\mathbf{X}_{j+1} &= \mathrm{MP}(\mathbf{X}_j, \mathbf{A}; \boldsymbol{\Theta}_{\mathrm{MP}}) \\ &= \mathrm{ReLU}\big(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}_j\mathbf{W} + \mathbf{X}_j\mathbf{V}\big) \end{aligned} \quad (1)$$

where $\boldsymbol{\Theta}_{\mathrm{MP}} = \{\mathbf{W} \in \mathbb{R}^{F_j\times F_{j+1}}, \mathbf{V} \in \mathbb{R}^{F_j\times F_{j+1}}\}$ are the trainable weights relative to the mixing and skip component of the layer, respectively. Several other types of MP (*e.g.*, those proposed in [11]–[16]) can seamlessly be used in conjunction with NDP pooling. In the presence of annotated edges, the MP operation can be extended by following [17] and [18].

III. Graph Coarsening With NDP

In this section, we describe the proposed NDP operation that consists of the three steps depicted in Fig. 1: (a) decimate the nodes by dropping one of the two sides of the MAXCUT partition; (b) connect the remaining nodes with a *link construction* procedure; (c) sparsify the adjacency matrix resulting from the coarsened Laplacian, so that only *strong* connections are kept, i.e., those edges whose weight is associated with an entry of the adjacency matrix above a given threshold ϵ. The proposed method is completely unsupervised and the coarsened graphs are precomputed before training the GNN.

A. Node Decimation With MAXCUT Spectral Partitioning

Similar to pooling operations in Convolutional Neural Networks (CNNs) that compute local summaries of neighboring pixels, we propose a pooling procedure that provides effective coverage of the whole graph and reduces the number of nodes approximately by a factor of 2. This can be achieved by partitioning nodes in two sets, so that nodes in one set are strongly connected to the complement nodes of the partition, and then dropping one of the two sets. The rationale is that strongly connected nodes exchange a lot of information after an MP operation and, as a result, they are highly dependent and their features become similar. Therefore, one set alone can represent the whole graph sufficiently well. This is similar to pooling in CNNs, where the maximum or the average is extracted from a small patch of neighboring pixels, which are assumed to be highly correlated and contain similar information. In the following, we formalize the problem of finding the optimal

subset of vertices that can be used to represent the whole graph.

The partition of the vertices (a cut) that maximizes the volume of edges whose endpoints are on opposite sides of the partition is the solution of the MAXCUT problem [19]. The MAXCUT objective is expressed by the integer quadratic problem

$$\max_{\mathbf{z}} \sum_{i,j\in\mathcal{V}} a_{ij}(1 - z_i z_j) \quad \text{s.t. } z_i \in \{-1, 1\} \tag{2}$$

where $\mathbf{z}$ is the vector containing the optimization variables z_i for $i = 1, \ldots, N$ indicating to which side of the bipartition the node i is assigned to; a_{ij} is the entry at row i and column j of $\mathbf{A}$. Problem (2) is NP-hard and heuristics must be considered to solve it. The heuristic that gives the best-known MAXCUT approximation in polynomial time is the Goemans-Williamson algorithm, which is based on the Semidefinite Programming (SDP) relaxation [20]. Solving SDP is cumbersome and requires specific optimization programs that scale poorly on large graphs. Therefore, we hereby propose a simple algorithm based on the Laplacian spectrum.

First, we rewrite the objective function in (2) as a quadratic form of the graph Laplacian

$$\begin{aligned}\sum_{i,j} a_{ij}(1 - z_i z_j) &= \sum_{i,j} a_{ij}\left(\frac{z_i^2 + z_j^2}{2} - z_i z_j\right) \\ &= \frac{1}{2}\sum_i \left[\sum_j a_{ij}\right] z_i^2 + \frac{1}{2}\sum_j \left[\sum_i a_{ij}\right] z_j^2 \\ &\quad - \sum_{i,j} a_{ij} z_i z_j \\ &= \frac{1}{2}\sum_i d_{ii} z_i^2 + \frac{1}{2}\sum_j d_{jj} z_j^2 - \mathbf{z}^T\mathbf{A}\mathbf{z} \\ &= \mathbf{z}^T\mathbf{D}\mathbf{z} - \mathbf{z}^T\mathbf{A}\mathbf{z} = \mathbf{z}^T\mathbf{L}\mathbf{z}.\end{aligned}$$

Then, we consider a continuous relaxation of the integer problem (2) by letting the discrete partition vector $\mathbf{z}$ assume continuous values, contained in a vector $\mathbf{c}$

$$\max_{\mathbf{c}} \ \mathbf{c}^T\mathbf{L}\mathbf{c}, \quad \text{s.t. } \mathbf{c} \in \mathbb{R}^N \text{ and } \|\mathbf{c}\|^2 = 1. \tag{3}$$

Equation 3 can be solved by considering the Lagrangian $\mathbf{c}^T\mathbf{L}\mathbf{c} + \lambda\mathbf{c}^T\mathbf{c}$ to find the maximum of $\mathbf{c}^T\mathbf{L}\mathbf{c}$ under constraint $\|\mathbf{c}\|^2 = 1$. By setting the gradient of the Lagrangian to zero, we recover the eigenvalue equation $\mathbf{L}\mathbf{c} + \lambda\mathbf{c} = 0$. All the eigenvalues of $\mathbf{L}$ are nonnegative and, by restricting the space of feasible solutions to vectors of the unitary norm, the trivial solution $\mathbf{c}^* = \infty$ is excluded. In particular, if $\|\mathbf{c}\|^2 = 1$, $\mathbf{c}^T\mathbf{L}\mathbf{c}$ is a Rayleigh quotient and reaches its maximum $\lambda_{\max}$ (the largest eigenvalue of $\mathbf{L}$) when $\mathbf{c}^*$ corresponds to $\mathbf{v}_{\max}$, the eigenvector associated with $\lambda_{\max}$.

Since the components of $\mathbf{v}_{\max}$ are real, we apply a rounding procedure to find a discrete solution. Specifically, we are looking for a partition $\mathbf{z}^* \in \mathcal{Z}$, where $\mathcal{Z} = \{\mathbf{z} : \mathbf{z} \in \{-1, 1\}^N\}$ is the set of all feasible cuts, so that $\mathbf{z}^*$ is the closest (in a least-square sense) to $\mathbf{c}^*$. This amounts to solving the problem

$$\mathbf{z}^* = \arg\min\{\|\mathbf{c}^* - \mathbf{z}\|_2 : \mathbf{z} \in \mathcal{Z}\} \tag{4}$$

with the optimum given by

$$z_i^* = \begin{cases} 1, & c_i^* \geq 0 \\ -1, & c_i^* < 0. \end{cases} \tag{5}$$

By means of the rounding procedure in (5), the nodes in $\mathcal{V}$ are partitioned in two sets, $\mathcal{V}^+$ and $\mathcal{V}^- = \mathcal{V} \setminus \mathcal{V}^+$, such that

$$\mathcal{V}^+ = \{i \in \mathcal{V} : \mathbf{v}_{\max}[i] \geq 0\}. \tag{6}$$

In the NDP algorithm, we always drop the nodes in $\mathcal{V}^-$, i.e., the nodes associated with a negative value in $\mathbf{v}_{\max}$. However, it would be equivalent to drop the nodes in $\mathcal{V}^+$. The node decimation procedure offers two important advantages: 1) it removes approximately half of the nodes when applied, i.e., $|\mathcal{V}^+| \approx |\mathcal{V}^-|$ and 2) the eigenvector $\mathbf{v}_{\max}$ can be quickly computed with the power method [21].

There exists an analogy between the proposed spectral algorithm for partitioning the graph nodes and spectral clustering [22]. However, spectral clustering solves a minCUT problem [23], [24], which is somehow orthogonal to the MAXCUT problem considered here. In particular, spectral clustering identifies $K \geq 2$ clusters of densely connected nodes by cutting the smallest volume of edges in the graph, while our algorithm cuts the largest volume of edges yielding two sets of nodes, $\mathcal{V}^+$ and $\mathcal{V}^-$, that cover the original graph in a similar way. Moreover, spectral clustering partitions the nodes in K clusters based on the values of the eigenvectors associated with the $M \geq 1$ smallest eigenvalues, while our algorithm partitions the nodes in two sets-based only on the last eigenvector $\mathbf{v}_{\max}$.

B. Links Construction on the Coarsened Graph

After dropping nodes in $\mathcal{V}^-$ and all their incident edges, the resulting graph is likely to be disconnected. Therefore, we use a link construction procedure to obtain a connected graph supported by the nodes in $\mathcal{V}^+$. Specifically, we adopt the Kron reduction [25] to generate a new Laplacian $\mathbf{L}^{(1)}$, which is computed as the Schur complement of $\mathbf{L}$ with respect to the nodes in $\mathcal{V}^-$. In detail, the reduced Laplacian $\mathbf{L}^{(1)}$ is

$$\mathbf{L}^{(1)} = \mathbf{L} \setminus \mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-} = \mathbf{L}_{\mathcal{V}^+,\mathcal{V}^+} - \mathbf{L}_{\mathcal{V}^+,\mathcal{V}^-}\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}^{-1}\mathbf{L}_{\mathcal{V}^-\mathcal{V}^+} \tag{7}$$

where $\mathbf{L}_{\mathcal{V}^+,\mathcal{V}^-}$ identifies a submatrix of $\mathbf{L}$ with rows (columns) corresponding to the nodes in $\mathcal{V}^+$ ($\mathcal{V}^-$).

It is possible to show that $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is always invertible if the associated adjacency matrix $\mathbf{A}$ is *irreducible*. We note that $\mathbf{A}$ is irreducible when the graph is not disconnected (i.e., has a single component), a property that holds when the algebraic multiplicity of the eigenvalue $\lambda_{\min} = 0$ is 1.

Let us consider the case where $\mathbf{A}$ has no self loops. The Laplacian is by definition a weakly diagonally dominant matrix, since $\mathbf{L}_{ii} = \sum_{j=1, j\neq i}^{N} |\mathbf{L}_{ij}|$ for all $i \in \mathcal{V}$. If $\mathbf{A}$ is irreducible, then $\mathbf{L}$ is also irreducible. This implies that the strict inequality $\mathbf{L}_{ii} > \sum_{j=1, j\neq i}^{n} |\mathbf{L}_{ij}|$ holds for at least one vertex $i \in \mathcal{V}^-$. It follows that the Kron-reduced Laplacian $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is also irreducible, diagonally dominant, and has at least one row with a strictly positive row sum. Hence, $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is invertible, as proven by [26] in Corollary 6.2.27. When $\mathbf{A}$ contains self-loops, the existence of the inverse of $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is still guaranteed through a small work-around, which is

discussed in Appendix A. Finally, if the graph is disconnected then $\mathbf{A}$ is reducible (i.e., it can be expressed in an upper-triangular block form by simultaneous row/column permutations); in this case, the Kron reduction can be computed by means of the generalized inverse $\mathbf{L}^{\dagger}_{\mathcal{V}^-,\mathcal{V}^-}$ and the solution corresponds to a generalized Schur complement of $\mathbf{L}$.

$\mathbf{L}^{(1)}$ in (7) is a well-defined Laplacian where two nodes are connected if and only if there is a path between them in $\mathbf{L}$ (connectivity preservation property). Also, $\mathbf{L}^{(1)}$ does not introduce self-loops and guarantees the preservation of resistance distance [8]. Finally, Kron reduction guarantees spectral interlacing between the original Laplacian $\mathbf{L} \in \mathbb{R}^{N\times N}$ and the new coarsened one $\mathbf{L}^{(1)} \in \mathbb{R}^{N_1\times N_1}$, with $N_1 \leq N$. Specifically, we have $\lambda_i \geq \lambda_i^{(1)} \geq \lambda_{N-N_1+i}$, $\forall i = 1,\dots,N_1$, where λ_i and $\lambda_i^{(1)}$ are the eigenvalues of $\mathbf{L}$ and $\mathbf{L}^{(1)}$, respectively.

The adjacency matrix of the new coarsened graph is recovered from the coarsened Laplacian

$$\mathbf{A}^{(1)} = \left(-\mathbf{L}^{(1)} + \operatorname{diag}\left(\left\{\sum_{j=1,j\neq i}^{N_1} \mathbf{L}^{(1)}_{ij}\right\}_{i=1}^{N_1}\right)\right). \tag{8}$$

We note that $\mathbf{A}^{(1)}$ does not contain self-loops; we refer to Appendix A for a discussion on how to handle the case with self-loops in the original adjacency matrix.

A pyramid of coarsened Laplacians is generated by recursively applying node decimation followed by Kron reduction. At the end of the procedure, the adjacency matrices $\mathcal{A} = \{\mathbf{A}^{(0)}, \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(l)}, \dots\}$ of the coarsened graphs are derived from the associated coarsened Laplacians. Note that we interchangeably refer with $\mathbf{A}^{(0)}$ or $\mathbf{A}$ to the original adjacency matrix. The adjacency matrices in $\mathcal{A}$ are used to implement hierarchical pooling in deep GNN architectures.

C. Graph Sparsification

To implement multiple pooling operations the graph must be coarsened several times. Due to the connectivity preservation property, by repeatedly applying Kron reduction the graph eventually becomes fully connected. This implies a high computational burden in deeper layers of the network, since the complexity of MP operations scales with the number of edges.

To address this issue, it is possible to apply the spectral sparsification algorithm proposed in [27] to obtain a sparser graph. However, we found that this procedure leads to numerical instability and poor convergence during the learning phase. Therefore, to limit the number of edges with nonzero weights we propose a sparsification procedure that removes from the adjacency matrix of the coarsened graph the edges whose weight is below a small user-defined threshold ϵ

$$\bar{\mathbf{A}}^{(i)} = \begin{cases} \bar{a}^{(l)}_{ij} = 0, & \text{if } \left|a^{(l)}_{ij}\right| \leq \epsilon \\ \bar{a}^{(l)}_{ij} = a^{(l)}_{ij}, & \text{otherwise.} \end{cases} \tag{9}$$

D. Pooling With Decimation Matrices

To pool the node features with a differentiable operation that can be applied while training the GNN, we multiply the graph signal $\mathbf{X}^{(l)}$ with a *decimation matrix* $\mathbf{S}^{(l)} \in \mathbb{N}^{N_{l+1}\times N_l}$. $\mathbf{S}^{(l)}$ is obtained by selecting from the identity matrix $\mathbf{I}_{N_l} \in \mathbb{N}^{N_l\times N_l}$ the rows corresponding to the vertices in $\mathcal{V}^+$

$$\mathbf{X}^{(l+1)} = \mathbf{S}^{(l)}\mathbf{X}^{(l)} = [\mathbf{I}_{N_l}]_{\mathcal{V}^+,:}\mathbf{X}^{(l)}. \tag{10}$$

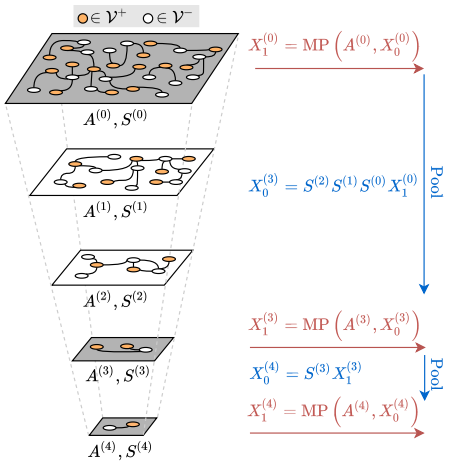

Fig. 2. This example shows how it is possible to skip some MP operations on intermediate levels of the pyramid of coarsened graphs. Such a procedure shares analogies with pooling with a larger stride in traditional CNNs and can be considered as a higher-order graph pooling. After the first MP operation on $\mathbf{A}^{(0)}$, the node features are pooled by applying in cascade three decimation matrices, $\mathbf{S}^{(0)}$, $\mathbf{S}^{(1)}$, and $\mathbf{S}^{(2)}$. Afterward, it is possible to directly perform an MP operation on $\mathbf{A}^{(3)}$, skipping the MP operations on $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$.

As discussed in Section III-A, NDP approximately halves the nodes of the current graph at each pooling stage. This is a consequence of the MAXCUT objective that splits the nodes into two sets so that the volume of edges crossing the partition, i.e., the edges to be cut, is maximized. Intuitively, if one of the two sets is much smaller than the other, more edges are cut by moving some nodes to the smaller set. For this reason, the application of a single decimation matrix $\mathbf{S}^{(l)}$ shares similarities with a classic pooling with stride 2 in CNNs.

It follows that a down-sampling ratio of $\approx 2^k$ can be obtained in NDP by applying k decimation matrices in cascade. This enables moving from level l to level $l+k$ ($k > 1$) in the pyramid of coarsened graphs. Fig. 2 shows an example of pooling with downsampling ratio ≈ 8, where the GNN performs MP on $\mathbf{A}^{(3)}$ right after $\mathbf{A}^{(0)}$.

IV. Analysis of the Graph Coarsening Procedure and Implementation Details

A. Numerical Precision in Eigendecomposition

The entries of $\mathbf{v}_{\max}$ associated with low-degree nodes assume very small values and their signs may be flipped due to numerical errors. The partition obtained by using $\mathbf{v}^s_{\max}$, i.e., the eigenvector of the symmetric Laplacian $\mathbf{L}_s$, in (6) is analytically the same. Indeed, since $\mathbf{v}^s_{\max} = \mathbf{D}^{-1/2}\mathbf{v}_{\max}$, the values of the two eigenvectors are rescaled by positive numbers and, therefore, the sign of their components is the same. However, a positive effect of the degree normalization is that the values in $\mathbf{v}^s_{\max}$ associated with nodes with a low degree are amplified.

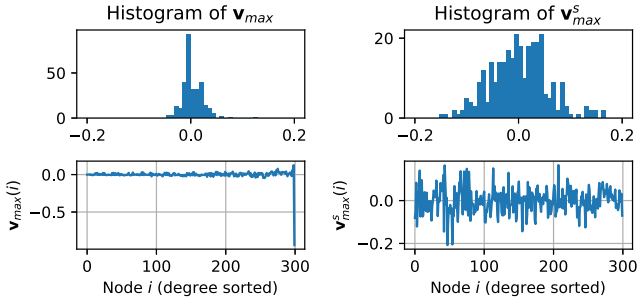

Fig. 3. (Left) Distribution and values assumed by $\mathbf{v}_{\max}$. (Right) Distribution and values assumed by $\mathbf{v}_{\max}$. The entries of the eigenvectors are sorted by node degree. A Stochastic Block Model graph was used in this example.

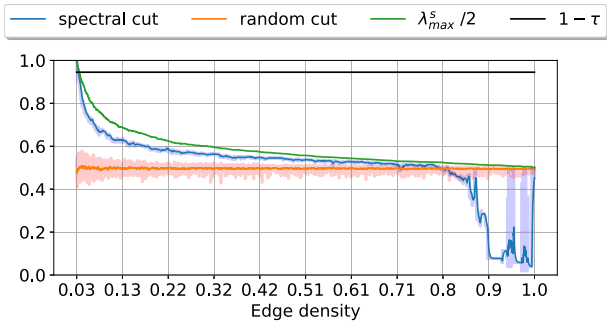

Fig. 4. Blue line: fraction of edges cut by the partition yield by the spectral algorithm. Orange line: fraction of edges removed by a random cut. Green line: the MAXCUT upper bound as a function of the largest eigenvalue $\lambda^s_{\max}$ of the symmetric Laplacian. Black line: the threshold from [28] indicating the value of $\lambda^s_{\max}/2$ below which one should switch to the random cut to obtain a solution guaranteed to be $\geq 0.53 \cdot$MAXCUT. The x-axis indicates the density of the graph connectivity, which increases by randomly adding edges.

Fig. 3 compares the values in the eigenvectors $\mathbf{v}_{\max}$ and $\mathbf{v}^s_{\max}$, computed on the same graph. Since many values in $\mathbf{v}_{\max}$ are concentrated around zero, partitioning the nodes according to the sign of the entries in $\mathbf{v}_{\max}$ gives numerically unstable results. On the other hand, since the values in $\mathbf{v}^s_{\max}$ are distributed more evenly the nodes can be partitioned more precisely.

Note that, even if the indexes of $\mathcal{V}^+$ are identified from the eigenvector of $\mathbf{L}_s$, the Kron reduction is still performed on the Laplacian $\mathbf{L}$. In the supplementary material, we report numerical differences in the size of the cut obtained on random graphs when using $\mathbf{v}_{\max}$ or $\mathbf{v}^s_{\max}$.

B. Evaluation of the Approximate MAXCUT Solution

Since computing the optimal MAXCUT solution is NP-hard, it is generally not possible to evaluate the quality of the cut found by the proposed spectral method (Section III-A) in terms of discrepancy from the MAXCUT. Therefore, to assess the quality of a solution we consider the following bounds:

$$0.5 \leq \frac{\text{MAXCUT}}{|\mathcal{E}|} \leq \frac{\lambda^s_{\max}}{2} \leq 1. \tag{11}$$

The value $\lambda^s_{\max}/2$ is an upper-bound of $\text{MAXCUT}/|\mathcal{E}|$, where $\lambda^s_{\max}$ is the largest eigenvalue of $\mathbf{L}_s$ and $|\mathcal{E}| = \sum_{i,j} a_{ij}$. The lower-bound 0.5 is given by the *random cut*, which uniformly assigns the nodes to the two sides of the partition.[1] The derivation of the upper-bound is in Appendix B.

To quantify the size of a cut induced by a partition vector $\mathbf{z}$, such as the one in (5), we introduce the function

$$0 \leq \gamma(\mathbf{z}) = \frac{\mathbf{z}^T \mathbf{L} \mathbf{z}}{2\sum_{i,j} a_{ij}} \leq \frac{\text{MAXCUT}}{|\mathcal{E}|} \tag{12}$$

which measures the proportion of edges cut by $\mathbf{z}$. Note that $\gamma(\cdot)$ depends also on $\mathbf{L}$, but we keep it implicit to simplify the notation.

Let us now consider the best- and worst case scenarios. The best case is the bipartite graph, where the MAXCUT is known and it cuts all the graph edges. The partition $\mathbf{z}$ found by our spectral algorithm on bipartite graphs is optimal, i.e., $\gamma(\mathbf{z}) = \text{MAXCUT}/|\mathcal{E}| = 1$. In graphs that are close to be bipartite or, in general, that have a very sparse and regular connectivity, a large percentage of edges can be cut if the nodes are partitioned correctly. Indeed, for these graphs, the MAXCUT is usually large and is closer to the upper-bound in (11). On the other hand, in very dense graphs the MAXCUT is smaller, as well as the gap between the upper- and lower-bound in (11). Notably, the worst case scenario is a complete graph where is not possible to cut more than half of the edges, i.e., MAXCUT $= 0.5$. We note that in graphs made of a sparse regular part that is weakly connected to a large dense part, the gaps in (11) can be arbitrarily large.

The proposed spectral algorithm is not designed to handle very dense graphs; an intuitive explanation is that $\mathbf{v}^s_{\max}$ can be interpreted as the graph signal with the highest frequency, since its sign oscillates as much as possible when transiting from a node to one of its neighbors. While such oscillation in the sign is clearly possible on bipartite graphs, in complete graphs it is not possible to find a signal that assumes an opposite sign on neighboring nodes, because all nodes are connected with each other. Remarkably, the solution (5) found by the spectral algorithm on very dense graphs can be worse than the random cut. A theoretical result found by Trevisan [28] states that a spectral algorithm, like the one we propose, is guaranteed to yield a cut larger than the random partition only when $\lambda^s_{\max} \geq 2(1-\tau) = 1.891$ (see Appendix C for details).

To illustrate how the size of the cut found by the spectral algorithm changes between the best- and worst case scenarios, we randomly add edges to a bipartite graph until it becomes complete. Fig. 4 illustrates how the size of the cut $\gamma(\mathbf{z})$ induced by the spectral partition $\mathbf{z}$ changes as more edges are added and the original structure of the graph is corrupted (blue line). The figure also reports the size of the random cut (orange line) and the MAXCUT upper bound from (12) (green line). The black line indicates the threshold from [28], i.e., the value of $\lambda^2_{\max}/2$ below which the spectral cut is no longer guaranteed to be larger than the random cut. The graph used to generate the figure is a regular grid; however, similar results hold also for other families of random graphs and are reported in the supplementary material.

Fig. 4 shows that the spectral algorithm finds a better-than-random cut even when $\lambda^s_{\max}/2 < 1-\tau$ (i.e., when the result

[1] A random cut $\mathbf{z}$ is, on average, at least 0.5 of the optimal cut $\mathbf{z}^*$: $\mathbb{E}[|\mathbf{z}|] = \sum_{(i,j)\in\mathcal{E}} \mathbb{E}[z_i z_j] = \sum_{(i,j)\in\mathcal{E}} \Pr[(i,j) \in \mathbf{z}] = (|\mathcal{E}|/2) \geq (|\mathbf{z}^*|/2)$.

Algorithm 1 Graph Coarsening

Input: adjacency matrix $\mathbf{A}$, coarsening levels $\mathcal{L}$, sparsification threshold ϵ

Output: coarsened adjacency matrices $\bar{\mathcal{A}}$, decimation matrices $\mathcal{S}$

1: $\mathbf{A}^{(0)} = \mathbf{A}$, $\mathbf{R} = \mathbf{I}_N$, $\mathcal{A} = \{\}$, $\mathcal{S} = \{\}$, $l = 0$
2: **while** $l \leq \max(\mathcal{L})$ **do**
3: $\quad \mathbf{A}^{(l+1)}, \mathbf{S}^{(l+1)} = \text{pool}(\mathbf{A}^{(l)})$
4: $\quad$ **if** $l \in \mathcal{L}$ **then**
5: $\quad\quad \mathcal{A} = \mathcal{A} \cup \mathbf{A}^{(l+1)}$, $\mathcal{S} = \mathcal{S} \cup \mathbf{S}^{(l+1)}\mathbf{R}$
6: $\quad\quad \mathbf{R} = \mathbf{I}_{N_l}$
7: $\quad$ **else**
8: $\quad\quad \mathbf{R} = \mathbf{S}^{(l+1)}\mathbf{R}$
9: $\quad l = l + 1$
10: $\bar{\mathcal{A}} = \{\bar{\mathbf{A}}^{(l)} : \bar{a}_{ij}^{(l)} = a_{ij}^{(l)}$ if $a_{ij}^{(l)} > \epsilon$ and 0 otherwise, $\forall \mathbf{A}^{(l)} \in \mathcal{A}\}$

Algorithm 2 pool$(\cdot)$ Function

Input: adjacency matrix $\mathbf{A}^{(l)} \in \mathbb{R}^{N_l \times N_l}$

Output: coarsened adjacency matrix $\mathbf{A}^{(l+1)} \in \mathbb{R}^{N_{l+1} \times N_{l+1}}$, decimation matrix $\mathbf{S}^{(l+1)} \in \mathbb{N}^{N_{l+1} \times N_l}$

1: get $\mathbf{L}^{(l)} = \mathbf{D}^{(l)} - \mathbf{A}^{(l)}$ and $\mathbf{L}_s^{(l)} = \mathbf{I} - (\mathbf{D}^{(l)})^{-\frac{1}{2}}\mathbf{A}^{(l)}(\mathbf{D}^{(l)})^{-\frac{1}{2}}$
2: compute the eigenvector $\mathbf{v}_{\max}^s$ of $\mathbf{L}_s^{(l)}$
3: partition vector $\mathbf{z}$ s.t. $z_i = 1$ if $\mathbf{v}_{\max}^s[i] \geq 0$, $z_i = -1$ if $\mathbf{v}_{\max}^s[i] < 0$
4: **if** $\gamma(\mathbf{z}) < 0.5$ **then**
5: $\quad$ random sample $z_i \sim \{-1, 1\}, \forall i = 1, \ldots, N_l$ (random cut)
6: $\mathcal{V}^+ = \{i : z_i = 1\}$, $\mathcal{V}^- = \{i : z_i = -1\}$
7: $\mathbf{L}^{(l+1)} = \mathbf{L}^{(l)} \setminus \mathbf{L}^{(l)}_{\mathcal{V}^-,\mathcal{V}^-}$ (Kron reduction)
8: $\mathbf{A}^{(l+1)} = -\mathbf{L}^{(l+1)} + \text{diag}(\sum_{j \neq i} \mathbf{L}_{ij}^{(l+1)})$
9: $\mathbf{S}^{(l+1)} = [\mathbf{I}_{N_{l+1}}]_{\mathcal{V}^+,:}$

from [28] does not hold), and only approaches the size of the random cut when the edge density is very high (70%–80%).

Importantly, when the size of the spectral partition becomes smaller than the random partition, the upper-bound $\lambda_{\max}^s/2 \approx 0.5$, meaning that the random cut is very close to the MAXCUT. To obtain a cut that is always at least as good as the random cut, we first compute the partition $\mathbf{z}$ as in (5) and evaluate its size $\gamma(\mathbf{z})$: if $\gamma(\mathbf{z}) < 0.5$, we return a random partition instead.

We conclude by noticing that, due to the smoothing effect of MP operations, the nodes belonging to densely connected graph components are likely to have very similar representations computed by the GNN; it is, therefore, not important which of these nodes are dropped by a random cut. The random cut in these cases not only is optimal in terms of the MAXCUT objective, but it also introduces stochasticity that provides robustness when training the GNN model.

C. Pseudocode

The procedure for generating the pyramid of coarsened adjacency matrices and the pooling matrices used for decimation is reported in Algorithm 1. $\mathcal{L}$ is a list of positive integers indicating the levels in the pyramid of coarsened Laplacians that we want to compute. For instance, given levels $\mathcal{L} = [1, 3, 5]$ for a graph of N nodes, the algorithm will return the coarsened graphs with approximately $N/2$, $N/8$, and $N/32$ nodes (in general, $N/2^{l_i}$ for each l_i in $\mathcal{L}$). Matrix $\mathbf{R}$ is a buffer that accumulates decimation matrices when one or more coarsening levels are skipped. This happens when the GNN implements high order pooling, as discussed in Section III-D.

Algorithm 2 shows the details of the pooling function, used in line 3 of Algorithm 1.

D. Computational Cost Analysis

The most expensive operations in the NDP algorithm are i) the cost of computing the eigenvector $\mathbf{v}_{\max}^s$, and ii) the cost of inverting the submatrix $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ within the Kron reduction.

Computing all eigenvectors has a cost $\mathcal{O}(N^3)$, where N is the number of nodes. However, computing only the eigenvector corresponding to the largest eigenvalue is fast when using the power method [29], which requires only few iterations (usually 5–10), each one of complexity $\mathcal{O}(N^2)$. The cost of inverting $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is $\mathcal{O}(|\mathcal{V}^-|^3)$, where $|\mathcal{V}^-|$ is the number of nodes that are dropped.

We notice that the coarsened graphs are precomputed before training the GNN. Therefore, the computational time of graph coarsening is much lower compared to training the GNN for several epochs, since each MP operation in the GNN has a cost $\mathcal{O}(N^2)$.

E. Structure of the Sparsified Graphs

When applying the sparsification, the spectrum of the resulting adjacency matrix $\bar{\mathbf{A}}$ is preserved, up to a small factor that depends on ϵ, with respect to the spectrum of $\mathbf{A}$.

Theorem 1: Let $\mathbf{Q}$ be a matrix used to remove small values in the adjacency matrix $\mathbf{A}$, which is defined as

$$\mathbf{Q} = \begin{cases} q_{ij} = -a_{ij}, & \text{if } |a_{ij}| \leq \epsilon \\ q_{ij} = 0, & \text{otherwise.} \end{cases} \tag{13}$$

Each eigenvalue $\bar{\alpha}_i$ of the sparsified adjacency matrix $\bar{\mathbf{A}} = \mathbf{A} + \mathbf{Q}$ is bounded by

$$\bar{\alpha}_i \leq \alpha_i + \mathbf{u}_i^T \mathbf{Q} \mathbf{u}_i \tag{14}$$

where α_i and $\mathbf{u}_i$ are eigenvalue-eigenvector pairs of $\mathbf{A}$.

Proof: Let $\mathbf{P}$ be a matrix with elements $p_{ij} = \text{sign}(q_{ij})$ and consider the perturbation $\mathbf{A} + \epsilon\mathbf{P}$, which modifies the eigenvalue problem $\mathbf{A}\mathbf{u}_i = \alpha_i \mathbf{u}_i$ in

$$(\mathbf{A} + \epsilon\mathbf{P})(\mathbf{u}_i + \mathbf{u}_\epsilon) = (\alpha_i + \alpha_\epsilon)(\mathbf{u}_i + \mathbf{u}_\epsilon) \tag{15}$$

where α_ϵ is a small number and $\mathbf{u}_\epsilon$ a small vector, which are unknown and indicate a perturbation on α_i and $\mathbf{u}_i$, respectively. By expanding (15), then canceling the equation $\mathbf{A}\mathbf{u}_i = \alpha_i \mathbf{u}_i$ and the high order terms $\mathcal{O}(\epsilon^2)$, one obtains

$$\mathbf{A}\mathbf{u}_\epsilon + \epsilon\mathbf{P}\mathbf{u}_i = \alpha_i \mathbf{u}_\epsilon + \alpha_\epsilon \mathbf{u}_i. \tag{16}$$

Since $\mathbf{A}$ is symmetric, its eigenvectors can be used as a basis to express the small vector $\mathbf{u}_\epsilon$

$$\mathbf{u}_\epsilon = \sum_{j=1}^{N} \delta_j \mathbf{u}_j \tag{17}$$

where δ_j are (small) unknown coefficients. Substituting (17) in (16) and bringing $\mathbf{A}$ inside the summation, gives

$$\sum_{j=1}^{N} \delta_j \mathbf{A}\mathbf{u}_j + \epsilon \mathbf{P}\mathbf{u}_i = \alpha_i \sum_{j=1}^{N} \delta_j \mathbf{u}_j + \alpha_\epsilon \mathbf{u}_i. \tag{18}$$

By considering the original eigenvalue problem that gives $\sum_{j=1}^{N} \delta_j \mathbf{A}\mathbf{u}_j = \sum_{j=1}^{N} \delta_j \alpha_j \mathbf{u}_j$ and by left-multiplying each term with $\mathbf{u}_i^T$, (18) becomes

$$\mathbf{u}_i^T \sum_{j=1}^{N} \delta_j \alpha_j \mathbf{u}_j + \mathbf{u}_i^T \epsilon \mathbf{P}\mathbf{u}_i = \mathbf{u}_i^T \alpha_i \sum_{j=1}^{N} \delta_j \mathbf{u}_j + \mathbf{u}_i^T \alpha_\epsilon \mathbf{u}_i. \tag{19}$$

Since eigenvectors are orthogonal, $\mathbf{u}_i^T \mathbf{u}_j = 0, \forall j \neq i$ and $\mathbf{u}_i^T \mathbf{u}_j = 1$, for $j = i$, (19) becomes

$$\begin{aligned} \mathbf{u}_i^T \delta_i \alpha_i \mathbf{u}_i + \mathbf{u}_i^T \epsilon \mathbf{P}\mathbf{u}_i &= \mathbf{u}_i^T \alpha_i \delta_i \mathbf{u}_i + \mathbf{u}_i^T \alpha_i \mathbf{u}_i \\ \mathbf{u}_i^T \epsilon \mathbf{P}\mathbf{u}_i &= \mathbf{u}_i^T \alpha_\epsilon \mathbf{u}_i = \alpha_\epsilon \end{aligned} \tag{20}$$

which, in turn, gives

$$\alpha_\epsilon = \mathbf{u}_i^T \epsilon \mathbf{P}\mathbf{u}_i \geq \mathbf{u}_i^T \mathbf{Q}\mathbf{u}_i \tag{21}$$

as $\mathbf{Q} \leq \epsilon \mathbf{P}$. ■

A common way to measure the similarity of two graphs is to compare the spectrum of their Laplacians. To extend the results of Theorem 1 to the spectra of the Laplacians $\mathbf{L}$ and $\bar{\mathbf{L}}$, respectively, associated with the original and sparsified adjacency matrices $\mathbf{A}$ and $\bar{\mathbf{A}}$, it is necessary to consider the relationships between the eigenvalues of $\mathbf{A}$ and $\mathbf{L}$. For a d-regular graph, the relationship $\lambda_i = d - \alpha_i$ links the ith eigenvalue λ_i of $\mathbf{L}$ to the ith eigenvalue α_i of $\mathbf{A}$ [30]. However, for a general graph it is only possible to derive a loose bound, $d_{\max} - \alpha_n \leq \lambda_n \leq d_{\max} - \alpha_1$, that depends on the maximum degree $d_{\max}$ of the graph [31, Lemma 2.21].

Therefore, we numerically compare the spectra of the Laplacians associated with the matrices in $\mathbf{A}$ and $\bar{\mathbf{A}}$. In particular, Fig. 5(*top-left*) depicts the spectrum of the Laplacian associated with the original graph $\mathbf{A}^{(0)}$ (black dotted line) and the spectra $\Lambda(\mathbf{L}^{(1)})$, $\Lambda(\mathbf{L}^{(2)})$, $\Lambda(\mathbf{L}^{(3)})$ of the Laplacians associated with $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$. Fig. 5(*top-right*) depicts the spectra of the Laplacians $\bar{\mathbf{L}}^{(1)}$, $\bar{\mathbf{L}}^{(2)}$, $\bar{\mathbf{L}}^{(3)}$ associated with the sparsified matrices $\bar{\mathbf{A}}^{(1)}$, $\bar{\mathbf{A}}^{(2)}$, and $\bar{\mathbf{A}}^{(3)}$. It is possible to observe that the spectra of $\mathbf{L}^{(l)}$ and $\bar{\mathbf{L}}^{(l)}$ are almost identical and therefore, to better visualize the differences, we show in Fig. 5(*bottom*) the absolute differences $|\Lambda(\mathbf{L}^{(l)}) - \Lambda(\bar{\mathbf{L}}^{(l)})|$. The graph used in Fig. 5 is a random sensor network and the sparsification threshold is $\epsilon = 10^{-2}$, which is the one adopted in all our experiments.

To quantify how much the coarsened graph changes as a function of ϵ, we consider the *spectral distance* that measures a dissimilarity between the spectra of the Laplacians associated with $\mathbf{A}$ and $\bar{\mathbf{A}}$ [32]. The spectral distance is computed as

$$SD(\mathbf{L}, \bar{\mathbf{L}}; \epsilon) = \frac{1}{K} \sum_{k=2}^{K+1} \frac{|\bar{\lambda}_k(\epsilon) - \lambda_k|}{\lambda_k} \tag{22}$$

where $\{\lambda_k\}_{k=2}^{K+1}$ and $\{\bar{\lambda}_k(\epsilon)\}_{k=2}^{K+1}$ are, respectively, the K smallest nonzero eigenvalues of $\mathbf{L}$ and $\bar{\mathbf{L}}$.

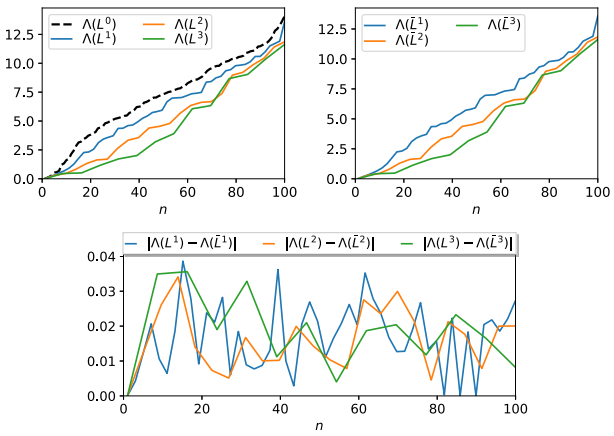

Fig. 5. *Top-left*: Spectrum of the Laplacians associated with the original adjacency $\mathbf{A}^{(0)}$ and the coarsened versions $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$ obtained with the NDP algorithm. *Top-right*: Spectrum of the Laplacians associated with the sparsified adjacency matrices $\bar{\mathbf{A}}^{(1)}$, $\bar{\mathbf{A}}^{(2)}$, and $\bar{\mathbf{A}}^{(3)}$. *Bottom*: Absolute difference between the spectra of the Laplacians.

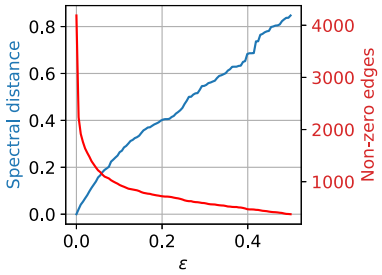

Fig. 6. In blue, the variation of spectral distance between the Laplacian $\mathbf{L}$ and the Laplacian $\bar{\mathbf{L}}$, associated with the adjacency matrix $\mathbf{A}$ sparsified with threshold ϵ. In red, the number of edges that remain in $\bar{\mathbf{L}}$.

Fig. 6 depicts in blue the variation of spectral distance between $\mathbf{L}$ and $\bar{\mathbf{L}}$, as we increase the threshold ϵ used to compute $\bar{\mathbf{A}}$. The red line indicates the number of edges that remain in $\bar{\mathbf{A}}$ after sparsification. It is possible to see that for small increments of ϵ the spectral distance increases linearly, while the number of edges in the graph drops exponentially. Therefore, with a small ϵ it is possible to discard a large amount of edges with minimal changes in the graph spectrum.

The graph used to generate Fig. 6 is a sensor network; results for other types of the graph are in the supplementary material.

V. Related Work on Graph Pooling

We discuss related work on GNN pooling by distinguishing between *topological* and *feature-based* pooling methods.

A. Topological Pooling Methods

Similar to NDP, topological pooling methods precompute coarsened graphs before training based on their topology. Topological pooling methods are usually unsupervised, as they define how to coarsen the graph outside of the learning procedure. The GNN is then trained to fit its node representations to these predetermined structures. Precomputing graph coarsening not only makes the training much faster by

avoiding to perform graph reduction at every forward pass, but it also provides a strong inductive bias that prevents degenerate solutions, such as entire graphs collapsing into a single node or entire graph sections being discarded. This is important when dealing with small data sets or, as we show in Section VI-B, in tasks such as graph signal classification.

The approach that is most related to NDP and that has been adopted in several GNN architectures to perform pooling [11], [33]–[36], consists of coarsening the graph with GRACLUS, a hierarchical spectral clustering algorithm [37]. At each level l, two vertices i and j are clustered together in a new vertex k. Then, a standard pooling operation (average or max pool) is applied to compute the node feature $\mathbf{x}_k^{(l+1)}$ from $\mathbf{x}_i^{(l)}$ and $\mathbf{x}_j^{(l)}$. This approach has several drawbacks. First, the connectivity of the original graph is not preserved in the coarsened graphs and the spectrum of their associated Laplacians is usually not contained in the spectrum of the original Laplacian. Second, GRACLUS pooling adds "fake" nodes so that they can be exactly halved at each pooling step; this not only injects noisy information in the graph signal, but also increases the computational complexity in the GNN. Finally, clustering depends on the initial ordering of the nodes, which hampers stability and reproducibility.

An alternative approach is to directly cluster the rows (or the columns) of the adjacency matrix, as done by approach proposed in [38], which decomposes $\mathbf{A}$ in two matrices $\mathbf{W} \in \mathbb{R}^{N\times K}$ and $\mathbf{H} \in \mathbb{R}^{K\times N}$ using the Nonnegative Matrix Factorization (NMF) $\mathbf{A} \approx \mathbf{WH}$. The NMF inherently clusters the columns of $\mathbf{A}$ since the minimization of the NMF objective is equivalent to the objective in k-means clustering [39]. In particular, $\mathbf{W}$ is interpreted as the cluster representatives matrix and $\mathbf{H}$ as a soft-cluster assignment matrix of the columns in $\mathbf{A}$. Therefore, the pooled node features and the coarsened graph can be obtained as $\mathbf{X}^{(1)} = \mathbf{H}^T\mathbf{X}$ and $\mathbf{A}^{(1)} = \mathbf{H}^T\mathbf{AH}$, respectively. The main drawback is that NMF does not scale well to large graphs.

B. Feature-Based Pooling Methods

These methods compute a coarsened version of the graph through differentiable functions, which are parametrized by weights that are optimized for the task at hand. Different from topological pooling, these methods account for the node features, which change as the GNN is trained. While this gives more flexibility in adapting the coarsening on the data and the task at hand, GNNs with feature-based pooling have more parameters; as such, training is slower and more difficult.

DiffPool [40] is a pooling method that learns differentiable soft assignments to cluster the nodes at each layer. DiffPool uses two MP layers in parallel: one to update the node features, and one to generate soft cluster assignments. The original adjacency matrix acts as a prior when learning the cluster assignments, while an entropy-based regularization encourages sparsity in the cluster assignments. The application of this method to large graphs is not practical, as the cluster assignment matrix is dense and its size is $N \times K$, where K is the number of nodes of the coarsened graph.

A second approach, dubbed *Top-K* pooling [41], [42], learns a projection vector that is applied to each node feature to obtain a score. The nodes with the K highest scores are retained, while the remaining ones are dropped. Since the top-K selection is not differentiable, the scores are also used as a gating for the node features, allowing gradients to flow through the projection vector during backpropagation. Top-K is more memory efficient than DiffPool as it avoids generating cluster assignments. A variant proposed in [43] introduces in Top-K pooling a soft attention mechanism for selecting the nodes to retain. Another variant of Top-K, called SAGPool, processes the node features with an additional MP layer before using them to compute the scores [44].

VI. Experiments

We consider two tasks on graph-structured data: graph classification and graph signal classification. The code used in all experiments is based on the Spektral library [45], and the code to replicate all experiments of this article is publicly available at GitHub.[2]

A. Graph Classification

In this task, the ith sample is a graph represented by the pair $\{\mathbf{A}_i, \mathbf{X}_i\}$ which must be classified with a label $\mathbf{y}_i$. We consider two synthetic data sets (*Bench-easy* and *Bench-hard*)[3] and eight data sets of real-world graphs: *Proteins*, *Enzymes*, *NCI1*, *MUTAG*, *Mutagenicity*, *D&D*, *COLLAB*, and *Reddit-Binary*.[4] When node features $\mathbf{X}$ are not available, we use node degrees and clustering coefficients as a surrogate. Moreover, we also use node labels as node features whenever they are available.

In the following, we compare NDP with GRACLUS [11], NMF [38], DiffPool [40], and Top-K [41]. In each experiment, we adopt a fixed network architecture, MP(32)-P(2)-MP(32)-P(2)-MP(32)-AvgPool-Softmax, where MP(32) stands for an MP layer as described in (1) configured with 32 hidden units and ReLU activations, P(2) is a pooling operation with stride 2, AvgPool is a global average pooling operation on all the remaining graph nodes, and Softmax indicates a dense layer with Softmax activation. As training algorithm, we use Adam [46] with initial learning rate 5e-4 and L_2 regularization with weight 5e-4. As an exception, for the Enzymes data set we used MP(64).

Additional baselines are the Weisfeiler–Lehman (WL) graph kernel [47], a GNN with only MP layers (*Flat*), and a network with only dense layers (*Dense*). The comparison with *Flat* helps to understand whether pooling operations are useful for a given task. The results obtained by *Dense*, instead, help to quantify how much additional information is brought by the graph structure compared to considering the node features alone. While recent graph kernels [48]–[50] and GNN architectures [51], [52] could be considered as further baselines for graph classification, the focus of our analysis and discussion is on graph pooling operators and, therefore, we point the interested reader toward the referenced articles.

[2] github.com/danielegrattarola/decimation-pooling

[3] https://github.com/FilippoMB/Benchmark_data set_for_graph_classification

[4] http://graphlearning.io

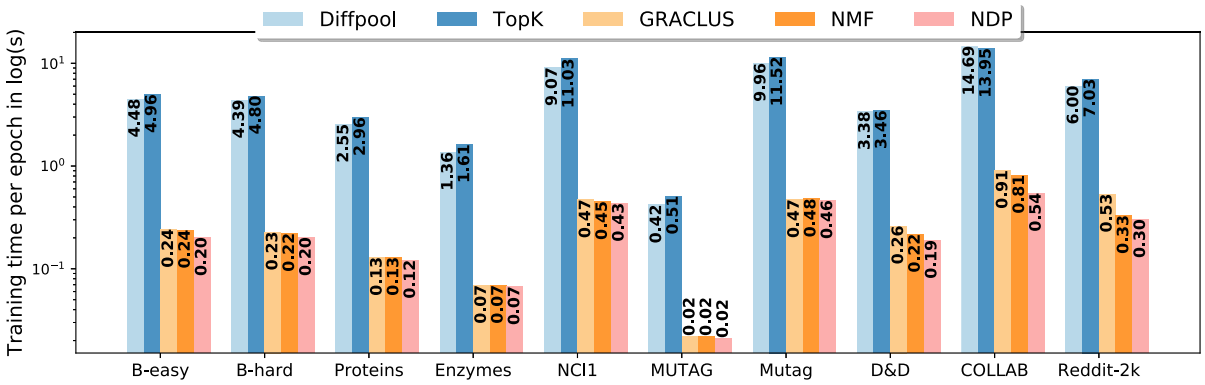

Fig. 7. Average training time per epoch (in seconds) for different pooling methods. The bar height is on logarithmic scale. Simulations were performed with an Nvidia RTX 2080 Ti.

TABLE I

GRAPH CLASSIFICATION ACCURACY. SIGNIFICANTLY BETTER RESULTS ($p < 0.05$) ARE IN BOLD

Dataset	WL	Dense	Flat	Diffpool	Top-K	GRACLUS	NMF	NDP
Bench-easy	92.6	29.3±0.3	98.5±0.3	**98.6±0.4**	82.4±8.9	97.5±0.5	97.4±0.8	97.4±0.9
Bench-hard	60.0	29.4±0.3	67.6±2.8	69.9±1.9	42.7±15.2	69.0±1.5	68.6±1.6	**71.9±0.8**
Proteins	71.2±2.6	68.7±3.3	72.6±4.8	72.8±3.5	69.6±3.3	70.3±2.6	71.6±4.1	**73.4±3.1**
Enzymes	33.6±4.1	45.7±9.9	**52.0±12.3**	24.6±5.3	31.4±6.8	42.0±6.7	39.9±3.6	44.5±7.4
NCI1	**81.1±1.6**	53.7±3.0	74.4±2.5	76.5±2.2	71.8±2.6	69.5±1.7	68.2±2.2	74.2±1.7
MUTAG	78.9±13.1	**91.1±7.1**	87.1±6.6	90.5±3.9	85.5±11.0	84.9±8.1	76.7±14.4	87.9±5.7
Mutagenicity	**81.7±1.1**	68.4±0.3	78.0±1.3	77.6±2.7	71.9±3.7	74.4±1.8	75.5±1.7	77.9±1.4
D&D	78.6±2.7	70.6±5.2	76.8±1.5	**79.3±2.4**	69.4±7.8	70.5±4.8	70.6±4.1	72.8±5.4
COLLAB	74.8±1.3	79.3±1.6	**82.1±1.8**	81.8±1.4	79.3±1.8	77.1±2.1	78.5±1.8	79.1±1.3
Reddit-Binary	68.2±1.7	48.5±2.6	80.3±2.6	86.8±2.1	74.7±4.5	79.2±0.4	52.0±2.1	**88.0±1.4**

To train the GNN on mini-batches of graphs with a variable number of nodes, we consider the disjoint union of the graphs in each mini-batch and train the GNN on the combined Laplacians and graph signals. See the supplementary material for an illustration.

We evaluate the model's performance by splitting the data set into ten folds. Each fold is, in turn, selected as the test set, while the remaining nine folds become the training set. For each different train/test split, we set aside 10% of the training data as a validation set, which is used for early stopping, i.e., we interrupt the training procedure after the loss on the validation set does not decrease for 50 epochs.

We report in Table I the test accuracy averaged over the ten folds. We note that no architecture outperforms every other in all tasks. The WL kernel achieves the best results on NCI1 and Mutagenicity, but it does not perform well on the other data sets. Interestingly, the *Dense* architecture achieves the best performance on MUTAG, indicating that in this case, the connectivity of the graphs does not carry useful information for the classification task. The performance of the *Flat* baseline indicates that in Enzymes and COLLAB pooling operations are not necessary to improve the classification accuracy.

NDP consistently achieves higher accuracy compared to GRACLUS and NMF, which are also topological pooling methods. We argue that the lower performance of GRACLUS is due to the fake nodes, which introduce noise in the graphs. Among the two feature-based pooling methods, DiffPool always outperforms Top-K. The reason is arguably

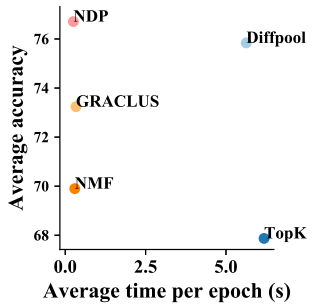

Fig. 8. Average training time per epoch against average accuracy, computed for each pooling method over the ten graph classification tasks.

that Top-K drops entire parts of the graphs, thus discarding important information for the classification [24], [43].

In Fig. 7, we report the training time for the five different pooling methods. As expected, GNNs configured with GRACLUS, NMF, and NDP are much faster to train compared to those based on DiffPool and Top$K$, with NDP being slightly faster than the other two topological methods. In Fig. 8, we plot the average training time per epoch against the average accuracy obtained by each pooling method on the ten data sets taken into account. The scatter plot is obtained from the data reported in Table I and Fig. 7. On average, NDP obtains the highest classification accuracy, slightly outperforming even Diffpool, while being, at the same time, the fastest among all pooling methods.

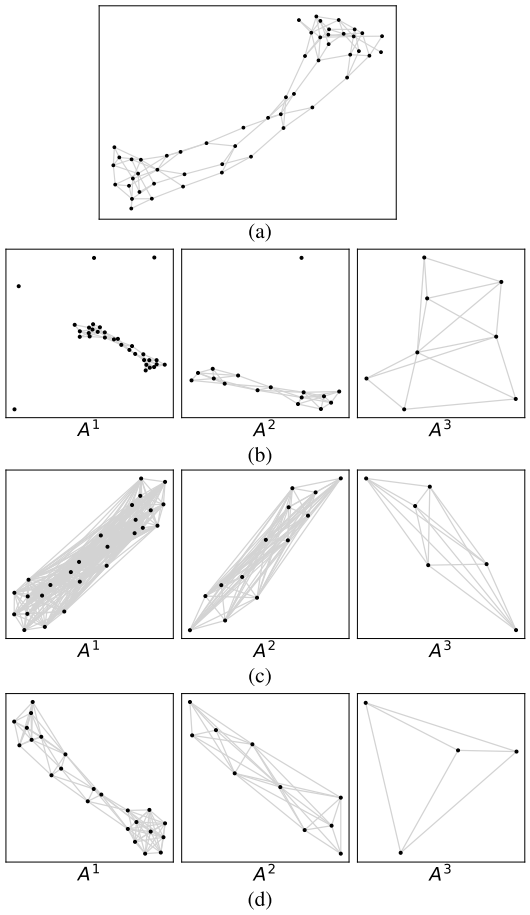

Fig. 9. Example of coarsening on one graph from the Proteins data set. In (a), the original adjacency matrix of the graph. In (b), (c), and (d) the edges of the Laplacians at coarsening levela 0, 1, and 2, as obtained by the three different pooling methods GRACLUS, NMF, and the proposed NDP.

To understand the differences between the topological pooling methods, we randomly selected one graph from the Proteins data set and show in Fig. 9 the coarsened graphs computed by GRACLUS, NMF, and NDP. From Fig. 9(b) we note that the graphs $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ in GRACLUS have additional nodes that are disconnected. As discussed in Section V, these are the fake nodes that are added to the graph so that their sizes can be halved at every pooling operation. Fig. 9(c) shows that NMF produces graphs that are very dense, as a consequence of the multiplication with the dense soft-assignment matrix to construct the coarsened graph. Finally, Fig. 9(d) shows that NDP produces coarsened graphs that are sparse and preserve well the topology of the original graph.

B. Graph Signal Classification

In this task, different graph signals $\mathbf{X}_i$, defined on the same adjacency matrix $\mathbf{A}$, must be classified with a label $\mathbf{y}_i$. We use the same architecture adopted for graph classification, with the only difference that each pooling operation is now implemented with stride 4: MP(32)-P(4)-MP(32)-P(4)-MP(32)-AvgPool-Softmax. We recall that when using NDP a stride of 4 is obtained by applying two decimation matrices in cascade, $\mathbf{S}^{(1)}\mathbf{S}^{(0)}$ and $\mathbf{S}^{(3)}\mathbf{S}^{(2)}$ (*cf.* Section III-D). We perform two graph signal classification experiments: image classification on MNIST and sentiment analysis on IMDB data set.

TABLE II
GRAPH SIGNAL CLASSIFICATION ACCURACY ON MNIST

DiffPool	Top-K	GRACLUS	NMF	NDP
24.00 $\pm$ 0.0	11.00 $\pm$ 0.0	96.21 $\pm$ 0.18	94.15 $\pm$ 0.17	**97.09** $\pm$ **0.01**

1) MNIST: For this experiment, we adopt the same settings described in [11]. To emulate a typical convolutional network operating on a regular 2-D grid, a 8-nearest neighbor (8-NN) graph is defined on the 28×28 pixels of the MNIST images, using as edge weights the following similarity score between nodes:

$$a_{ij} = \exp\left(-\frac{\|p_i - p_j\|^2}{\sigma^2}\right) \tag{23}$$

where p_i and p_j are the 2-D coordinates of pixel i and j. The graph signal $\mathbf{X}_i \in \mathbb{R}^{784\times 1}$ is the ith vectorized image.

Table II reports the average results achieved over ten independent runs by a GNN implemented with different pooling operators. Contrarily to graph classification, DiffPool and Top$K$ fail to solve this task and achieve an accuracy comparable to random guessing. On the contrary, the topological pooling methods obtain an accuracy close to a classical CNN, with NDP significantly outperforming the other two techniques.

We argue that the poor performance of the two feature-based pooling methods is attributable to 1) the low information content in the node features and 2) a graph that has a regular structure and is connected only locally. This means that the graph has a very large diameter (maximum shortest path), where information propagates slowly through MP layers. Therefore, even after MP, nodes in very different parts of the graph will end up having similar (if not identical) features, which leads to feature-based pooling methods to assign them to the same cluster. As a result, the graph collapses, becoming densely connected and losing its original structure. On the other hand, topological pooling methods can preserve the graph structure by operating on the whole adjacency matrix at once to compute the coarsened graphs and are not affected by uninformative node features.

2) IMDB: We consider the IMDB sentiment analysis data set of movies reviews, which must be classified as positive or negative. We use a graph that encodes the similarity of all words in the vocabulary. Each graph signal represents a review and consists of a binary vector with a size equal to the vocabulary, which assumes value 1 in correspondence of a word that appears at least once in the review, and 0 otherwise.

The graph is built as follows. First, we extract a vocabulary from the most common words in the reviews. For each review, we consider at most 256 words, padding with a special token the reviews that are shorter and truncating those that are longer. Then, we train a simple classifier consisting of a word embedding layer [53] of size 200, followed by a dense layer with a ReLU activation, a dropout layer [54] with probability 0.5, and a dense layer with sigmoid activation. After training, we extract the embedding vector of each word

TABLE III
GRAPH SIGNAL CLASSIFICATION ACCURACY ON IMDB SENTIMENT ANALYSIS DATA SET

# Words	Dense	LSTM	TCN	DiffPool	Top-K	GRACLUS	NMF	NDP
1k	82.65$\pm$0.01	86.58$\pm$0.03	85.61$\pm$0.14	50.00$\pm$0.0	50.00$\pm$0.0	85.03$\pm$0.10	82.51$\pm$0.11	**85.77$\pm$0.03**
5k	86.26$\pm$0.03	86.59$\pm$0.06	87.42$\pm$0.09	50.00$\pm$0.0	50.00$\pm$0.0	87.55$\pm$0.15	85.66$\pm$0.11	**87.79$\pm$0.02**
10k	83.75$\pm$0.02	85.98$\pm$0.04	87.38$\pm$0.07	50.00$\pm$0.0	50.00$\pm$0.0	87.29$\pm$0.07	OOM	**87.82$\pm$0.02**

in the vocabulary and construct a 4-NN graph, according to the Euclidean similarity between the embedding vectors.

As baselines, we consider the network used to generate the word embeddings (*Dense*) and two more advanced architectures. The first [*long-short term memory* (*LSTM*)], is a network where the dense hidden layer is replaced by an LSTM layer [55], which allows capturing the temporal dependencies in the sequence of words in the review. The other baseline [*temporal convolutional network* (*TCN*)] is a network where the hidden layers are 1-D convolutions with different dilation rates [56]. In particular, we used a Temporal Convolution Network [57] with seven residual blocks with dilations [1, 2, 4, 8, 16, 32, 64], kernel size 6, causal padding, and dropout probability 0.3. The results averaged over ten runs for vocabularies of different sizes (# Words) are reported in Table III.

Similar to the MNIST experiment, we notice that neither DiffPool nor Top$K$ are able to solve this graph signal classification task. The reason can be once again attributed to the low information content of the individual node features and in the sparsity of the graph signal (most node features are 0), which makes it difficult for the feature-based pooling methods to infer global properties of the graph by looking at local substructures.

On the other hand, NDP consistently outperforms the baselines, GRACLUS, and NMF. The coarsened graphs generated by NMF when the vocabulary has 10k words are too dense to fit in the memory of the GPU (Nvidia GeForce RTX 2080). Interestingly, the GNNs configured with GRACLUS and NDP always achieve better results than the *Dense* network, even if the latter generates the word embeddings used to build the graph on which the GNN operates. This can be explained by the fact that the *Dense* network immediately overfits the data set, whereas the graph structure provides a strong regularization, as the GNN combines only words that are neighboring on the vocabulary graph.

The *LSTM* baseline generally achieves better accuracy than *Dense*, since it captures the sequential ordering of the words in the reviews, which also helps to prevent overfitting on training data. Finally, the *TCN* baseline always outperforms *LSTM*, both in terms of accuracy and computational costs. This substantiates recent findings showing that convolutional architectures may be more suitable than recurrent ones for tasks involving sequential data [57].

VII. CONCLUSION

We proposed NDP, a pooling strategy for GNNs that reduces a graph based on properties of its Laplacian. NDP partitions the nodes into two disjoint sets by optimizing a MAXCUT objective. The nodes of one set are dropped, while the others are connected with Kron reduction to form a new smaller graph. Since Kron reduction yields dense graphs, a sparsification procedure is used to remove the weaker connections.

The algorithm we proposed to approximate the MAXCUT solution is theoretically grounded on graph spectral theory and achieves good results while being, at the same time, simple and efficient to implement. To evaluate the MAXCUT solution, we considered theoretical bounds and we introduced an experimental framework to empirically assess the quality of the solution.

We demonstrated that the graph sparsification procedure proposed in this work preserves the spectrum of the graph up to an arbitrarily small constant. In particular, we first derived an analytical relationship between the eigenvalues of the adjacency matrix of the original and sparsified graphs. Then, we performed numerical experiments to study how much the spectrum of the graph Laplacian varies, in practice, after sparsification.

We compared NDP with two main families of pooling methods for GNNs: topological (to which NDP belongs) and feature-based methods. NDP has advantages compared to both types of pooling. In particular, experimental results showed that NDP is computationally cheaper (in terms of both time and memory) than feature-based methods, while it achieves competitive performance on all the downstream tasks taken into account. An important finding in our results indicates that topological methods are the only viable approach in graph signal classification tasks.

APPENDIX

A. Kron Reduction in Graph With Self-Loops

If $\mathbf{A}$ contains self loops, the existence of the strict inequality condition $\mathbf{L}_{ii} > \sum_{j=1, j\neq i}^{n} |\mathbf{L}_{ij}|$ discussed in Section III-B is no more guaranteed. However, it is sufficient to consider the loopy-Laplacian $\mathbf{Q} = \mathbf{D} - \mathbf{A} + 2\text{diag}(\mathbf{A})$, where $\text{diag}(\mathbf{A})$ is the diagonal of $\mathbf{A}$, defined as $\{\mathbf{A}_{ii}\}_{i=1}^{N}$. $\mathbf{Q}$ is now an irreducible matrix and $\mathbf{Q}_{ii} > \sum_{j=1, j\neq i}^{n} |\mathbf{Q}_{ij}| + \mathbf{A}_{ii}$ holds for at least one vertex $i \in \mathcal{V}^{+}$. We notice that the adjacency matrix can be univocally recovered: $\mathbf{A} = -\mathbf{Q} + \text{diag}(\{\sum_{j=1, j\neq i}^{N} \mathbf{Q}_{ij}\}_{i=1}^{N})$. Therefore, from the Kron reduction $\mathbf{Q}^{(1)}$ of $\mathbf{Q}$ we can first recover $\mathbf{A}^{(1)}$ and then compute the reduced Laplacian as $\mathbf{L}^{(1)} = \mathbf{D}^{(1)} - \mathbf{A}^{(1)}$.

B. Derivation of the MAXCUT Upperbound

Let us consider the Rayleigh quotient

$$r(\mathbf{z}, \mathbf{L}) = \frac{\mathbf{z}^T \mathbf{L} \mathbf{z}}{\mathbf{z}^T \mathbf{z}} \tag{24}$$

which assumes its maximum value $\lambda_{\max}$ when $\mathbf{z}$ is the largest eigenvector of the Laplacian $\mathbf{L}$. When $\mathbf{z}$ is the partition vector

in (5), we have $r(\mathbf{z}, \mathbf{L}) \leq \lambda_{\max}$. As shown in Section III-A, the numerator in (24) can be rewritten as $\mathbf{z}^T\mathbf{L}\mathbf{z} = \sum_{i,j\in\mathcal{E}} a_{ij}(z_i - z_j)^2 = \sum_{i,j\in\mathcal{V}^+} a_{ij}(z_i - z_j)^2 + \sum_{i,j\in\mathcal{V}^-} a_{ij}(z_i - z_j)^2 + \sum_{i\in\mathcal{V}^+, j\in\mathcal{V}^-} a_{ij}(z_i - z_j)^2 = 0 + 0 + \sum_{i\in\mathcal{V}^+, j\in\mathcal{V}^-} a_{ij}2^2 = 4 \cdot \text{cut}(\mathbf{z})$, since $z_i = 1$ if $i \in \mathcal{V}^+$ and $z_i = -1$ if $i \in \mathcal{V}^-$ according to (5), and where cut($\mathbf{z}$) is the volume of edges crossing the partition induced by $\mathbf{z}$. From (5) also follows that the denominator in (24) is $\mathbf{z}^T\mathbf{z} = N$, since $z_i^2 = 1, \forall i$. By combining the results, we obtain

$$\frac{4 \cdot \text{cut}(\mathbf{z})}{N} \leq \lambda_{\max} \quad \forall \mathbf{z} \in \mathbb{R}^N \rightarrow \texttt{MAXCUT} \leq \lambda_{\max}\frac{N}{4}. \qquad (25)$$

When considering the symmetric Laplacian $\mathbf{L}_s$, we multiply (24) on both sides by $\mathbf{D}^{-1/2}$, changing the denominator into $\mathbf{z}^T\mathbf{D}\mathbf{z} = \sum_{i,i} d_{ii}z_i^2 = 2|\mathcal{E}|$. Replacing in (25) N with $2|\mathcal{E}|$ and $\lambda_{\max}$ with $\lambda^s_{\max}$, we get the bound $\texttt{MAXCUT}/|\mathcal{E}| \leq \lambda^s_{\max}/2$.

C. Relationship With Trevisan [28] Spectral Algorithm

The main result in [28] states that if $\lambda^s_{\max} \geq 2(1-\tau)$, then there exist a set of vertices $\mathcal{V}$ and a partition $(\mathcal{V}^1, \mathcal{V}^2)$ of $\mathcal{V}$ so that $|e(\mathcal{V}^1, \mathcal{V}^2)| \geq (1/2)(1\sqrt{16\tau})\text{vol}(\mathcal{V})$, where $\text{vol}(\mathcal{V}) = \sum_{i\in\mathcal{V}} d_i$ and $e(\mathcal{V}^1, \mathcal{V}^2)$ are the edges with one endpoint in $\mathcal{V}^1$ and the other in $\mathcal{V}^2$. In cases where an optimal solution cuts $1-\tau$ fraction of the edges, a partition found by a recursive spectral algorithm will remove $1 - 4\sqrt{\tau} + 8\tau$ of the edges. The optimal τ is value 0.0549 for which $((1 - 4\sqrt{\tau} + 8\tau)/(1-\tau))$ reaches its minimum 0.5311. When the largest eigenvalue $\lambda^s_{\max}$ is too small, the expected random cut is larger than the solution found by the spectral algorithm. The analysis in [28] shows that the spectral cut is guaranteed to be larger than the random cut only when $\lambda^s_{\max} \geq 2(1-\tau)$, i.e., when $\lambda^s_{\max} \geq 1.891$ given the optimal value $\tau = 0.0549$. Therefore, an algorithm that recursively cuts a fraction of edges according to the values in $\mathbf{v}^s_{\max}$ until $\lambda^s_{\max} \geq 2(1-\tau)$ and then performs a random cut, finds a solution that is always ≥ 0.5311 MAXCUT.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers for critically reading the manuscript and for giving important suggestions, which allowed them to significantly improve their work.

REFERENCES

[1] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[2] Z. Zhang, D. Chen, Z. Wang, H. Li, L. Bai, and E. R. Hancock, "Depth-based subgraph convolutional auto-encoder for network representation learning," *Pattern Recognit.*, vol. 90, pp. 363–376, Jun. 2019.

[3] L. Bai, L. Cui, S. Wu, Y. Jiao, and E. R. Hancock, "Learning vertex convolutional networks for graph classification," 2019, *arXiv:1902.09936*. [Online]. Available: http://arxiv.org/abs/1902.09936

[4] L. Bai, L. Cui, X. Bai, and E. R. Hancock, "Deep depth-based representations of graphs through deep learning networks," *Neurocomputing*, vol. 336, pp. 3–12, Apr. 2019.

[5] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.

[6] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1263–1272.

[7] N. Tremblay, P. Goncalves, and P. Borgnat, "Design of graph filters and filterbanks," in *Cooperative and Graph Signal Processing*. Amsterdam, The Netherlands: Elsevier, 2018, pp. 299–324.

[8] D. I. Shuman, M. J. Faraji, and P. Vandergheynst, "A multiscale pyramid transform for graph signals," *IEEE Trans. Signal Process.*, vol. 64, no. 8, pp. 2119–2134, Apr. 2016.

[9] F. Chung, "Laplacians and the cheeger inequality for directed graphs," *Ann. Combinatorics*, vol. 9, no. 1, pp. 1–19, Apr. 2005.

[10] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[13] P. Veli**v**ković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*. [Online]. Available: http://arxiv.org/abs/1710.10903

[14] F. Maria Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional ARMA filters," 2019, *arXiv:1901.01343*. [Online]. Available: http://arxiv.org/abs/1901.01343

[15] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2018, *arXiv:1810.00826*. [Online]. Available: http://arxiv.org/abs/1810.00826

[16] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[17] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3693–3702.

[18] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Proc. Eur. Semantic Web Conf.* Cham, Switzerland: Springer, 2018, pp. 593–607.

[19] L. Palagi, V. Piccialli, F. Rendl, G. Rinaldi, and A. Wiegele, "Computational approaches to max-cut," in *Handbook on Semidefinite, Conic and Polynomial Optimization*. Cham, Switzerland: Springer, 2012, pp. 821–847.

[20] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *J. ACM*, vol. 42, no. 6, pp. 1115–1145, Nov. 1995.

[21] F. M. Bianchi, E. Maiorino, L. Livi, A. Rizzi, and A. Sadeghian, "An agent-based algorithm exploiting multiple local dissimilarities for clusters mining and knowledge discovery," *Soft Comput.*, vol. 21, no. 5, pp. 1347–1369, Mar. 2017.

[22] U. von Luxburg, "A tutorial on spectral clustering," *Statist. Comput.*, vol. 17, no. 4, pp. 395–416, Dec. 2007.

[23] J. Shi and J. Malik, "Normalized cuts and image segmentation," Departmental Papers (CIS). 107, 2000.

[24] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 2729–2738.

[25] F. Dorfler and F. Bullo, "Kron reduction of graphs with applications to electrical networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 1, pp. 150–163, Jan. 2013.

[26] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[27] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng, "Spectral sparsification of graphs: Theory and algorithms," *Commun. ACM*, vol. 56, no. 8, pp. 87–94, 2013.

[28] L. Trevisan, "Max cut and the smallest eigenvalue," *SIAM J. Comput.*, vol. 41, no. 6, pp. 1769–1786, Jan. 2012.

[29] D. S. Watkins, *Fundamentals Matrix Computations*. vol. 64. Hoboken, NJ, USA: Wiley, 2004.

[30] J. F. Lutzeyer and A. T. Walden, "Comparing graph spectra of adjacency and Laplacian matrices," 2017, *arXiv:1712.03769*. [Online]. Available: http://arxiv.org/abs/1712.03769

[31] P. Zumstein, "Comparison of spectral methods through the adjacency matrix and the Laplacian of a graph," Diploma thesis, ETH Zürich, Zürich, Germany, 2005.

[32] A. Loukas, "Graph reduction with spectral and cut guarantees," *J. Mach. Learn. Res.*, vol. 20, no. 116, pp. 1–42, 2019.

[33] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*. [Online]. Available: http://arxiv.org/abs/1312.6203

[34] 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 *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5115–5124.

[35] M. Fey, J. E. Lenssen, F. Weichert, and H. Muller, "SplineCNN: Fast geometric deep learning with continuous B-spline kernels," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 869–877.

[36] R. Levie, M. Federico, X. Bresson, and B. Xavier, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019.

[37] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: Spectral clustering and normalized cuts," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 551–556.

[38] D. Bacciu and L. Di Sotto, "A non-negative factorization approach to node pooling in graph convolutional neural networks," in *Proc. 18th Int. Conf. Italian Assoc. Artif. Intell.*, 2019, pp. 294–306.

[39] C. Ding, X. He, and H. D. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2005, pp. 606–610.

[40] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," 2018, *arXiv:1806.08804*. [Online]. Available: http://arxiv.org/abs/1806.08804

[41] S. J. Hongyang Gao, "Graph U-Nets," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 2083–2092.

[42] C. Cangea, P. Velicković, N. Jovanović, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," in *Proc. Adv. Neural Inf. Process. Syst., Represent. Learn. Workshop (NeurIPS)*, 2018.

[43] B. Knyazev, G. W. Taylor, and M. Amer, "Understanding attention and generalization in graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, Inc., 2019, pp. 4202–4212. [Online]. Available: http://papers.nips.cc/paper/8673-understanding-attention-and-generalization-in-graph-neural-networks.pdf

[44] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. 36th Int. Conf. Mach. Learn.*, Jun. 2019, pp. 3734–3743.

[45] D. Grattarola and C. Alippi, "Graph neural networks in TensorFlow and keras with spektral," 2020, *arXiv:2006.12138*. [Online]. Available: http://arxiv.org/abs/2006.12138

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[47] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, Sep. 2011.

[48] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2015, pp. 1365–1374.

[49] A. Martino, A. Giuliani, and A. Rizzi, "(Hyper) graph embedding and classification via simplicial complexes," *Algorithms*, vol. 12, no. 11, p. 223, 2019.

[50] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein Weisfeiler-Lehman graph kernels," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 6439–6449.

[51] L. Bai, L. Cui, Y. Jiao, L. Rossi, and E. Hancock, "Learning backtrackless aligned-spatial graph convolutional networks for graph classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Jul. 24, 2020, doi: 10.1109/TPAMI.2020.3011866.

[52] J. Jiang, C. Xu, Z. Cui, T. Zhang, W. Zheng, and J. Yang, "Walk-steered convolution for graph classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4553–4566, Nov. 2020.

[53] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[55] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[56] A. van den Oord *et al.*, "WaveNet: A generative model for raw audio," 2016, *arXiv:1609.03499*. [Online]. Available: http://arxiv.org/abs/1609.03499

[57] S. Bai, J. Zico Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018, *arXiv:1803.01271*. [Online]. Available: http://arxiv.org/abs/1803.01271

Filippo Maria Bianchi received the Ph.D. degree from the Department of Information Engineering, Electronics, and Telecommunications, Sapienza University of Rome, Rome, Italy, in 2016.

He worked with Ryerson University, Toronto, ON, Canada, Universita' della Svizzera italiana, Lugano, Switzerland, and University of Pisa, Pisa, Italy. He is currently an Associate Professor with the Department of Mathematics and Statistics, UiT The Arctic University of Norway, Tromsø, Norway, and a Research Scientist with NORCE, Bergen, Norway. His research focuses on machine learning, complex networks, and dynamical systems.

Daniele Grattarola (Student Member, IEEE) received the M.Sc. degree *magna cum laude* in computer science and engineering from Politecnico di Milan, Milan, Italy, in 2017. He is currently pursuing the Ph.D. degree with the Faculty of Informatics, Universita' della Svizzera italiana, Lugano, Switzerland.

His research focuses on graph neural networks and their application to systems that change over time.

Lorenzo Livi (Member, IEEE) received the Ph.D. degree from the Department of Information Engineering, Electronics, and Telecommunications, Sapienza University of Rome, Rome, Italy, in 2014.

From January 2014 to April 2016, he was a Post-Doctoral Fellow with Ryerson University, Toronto, ON, Canada. From May 2016 to September 2016, he was a Post-Doctoral Fellow with the Politecnico di Milan, Milan, Italy and Universita' della Svizzera Italiana, Lugano, Switzerland. He is currently an Assistant Professor jointly appointed with the Departments of Computer Science and Mathematics, University of Manitoba, Winnipeg, MB, Canada. He is also a Lecturer (Assistant Professor) in Data Science with the Department of Computer Science, University of Exeter, Exeter, U.K. His research interests include machine learning, time series analysis and complex dynamical systems, with focused applications in systems biology and computational neuroscience.

Dr. Livi was awarded the Prestigious Tier 2 Canada Research Chair in Complex Data in November 2018. He is an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (TNNLS) and *Applied Soft Computing* (Elsevier).

Cesare Alippi (Fellow, IEEE) is a Professor with the Politecnico di Milan, Milan, Italy and Universita' della Svizzera italiana, Lugano, Switzerland. He is a Visiting Professor with the University of Guangzhou, Guangzhou, China and a Consultant Professor with the Northwestern Polytechnic, Xi'an, China. Current research activity addresses adaptation and learning in nonstationary environments, graph learning and Intelligence for embedded, IoT and cyber-physical systems.

Dr. Alippi is a member of the Administrative Committee of the IEEE Computational Intelligence Society, the Board of Governors Member of the International Neural Network Society, the Board of Directors Member of the European Neural Network Society, the Past Vice-President Education of the IEEE Computational Intelligence Society, the Past Associate Editor of the IEEE TRANSACTIONS ON EMERGING topics in computational intelligence, the *IEEE Computational Intelligence Magazine*, the IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENTS, the IEEE TRANSACTIONS ON NEURAL NETWORKS (and Learning Systems). He received the 2018 IEEE CIS Outstanding Computational Intelligence Magazine Award, the 2016 Gabor Award from the International Neural Networks Society and the IEEE Computational Intelligence Society Outstanding Transactions on Neural Networks and Learning Systems Paper Award; in 2013 the IBM Faculty Award; in 2004 the IEEE Instrumentation and Measurement Society Young Engineer Award.