Understanding Pooling in Graph Neural Networks

Daniele Grattarola, Daniele Zambon, Member, IEEE, Filippo Maria Bianchi, and Cesare Alippi, Fellow, IEEE

Abstract—Many recent works in the field of graph machine learning have introduced pooling operators to reduce the size of graphs. In this article, we present an operational framework to unify this vast and diverse literature by describing pooling operators as the combination of three functions: selection, reduction, and connection (SRC). We then introduce a taxonomy of pooling operators, based on some of their key characteristics and implementation differences under the SRC framework. Finally, we propose three criteria to evaluate the performance of pooling operators and use them to investigate the behavior of different operators on a variety of tasks.

Index Terms—Dimensionality reduction, graph neural networks (GNNs).

I. INTRODUCTION

Graph neural networks (GNNs) are often built by alternating layers that learn a transformation of the node features and pooling layers that reduce the number of nodes, similar to convolutional and pooling layers in convolutional neural networks (CNNs). Graph pooling can also be used as an independent operation to produce coarsened representations of given graphs.

While techniques for learning node representations have been largely studied, and different works have introduced general frameworks to unify the existing literature [1], [2], less attention has been devoted to pooling operators. Only a few recent works have attempted to systematically analyze the effect of pooling in GNNs [3], [4], and notably, a unifying formulation of pooling operators is still missing. In this article, we advance the understanding of graph pooling operators by proposing a universal and modular formalism to study pooling in GNNs, and we show which types of operators are more suitable for specific tasks.

The main contributions of this article can be summarized as follows.

1) We show that graph pooling operators can be seen as the combination of three functions: selection, reduction, and connection (SRC). The selection function groups the nodes of the input graph into subsets called supernodes; then, the reduction function aggregates each supernode to form an output node and its attributes; finally, the connection function links the reduced nodes with (possibly attributed) edges and outputs the pooled graph. The process is summarized in Fig. 1.

2) We propose a comprehensive taxonomy of pooling operators based on specific properties of the SRC functions. In particular, we identify four main properties that characterize pooling operators: 1) whether the SRC functions are learned or not; 2) whether their complexity is linear or quadratic in the number of nodes; 3) whether they produce graphs with a fixed or variable number of nodes; and 4) whether they pool the graphs hierarchically or globally.

3) We show that the three SRC functions can be interpreted as node- and graph-level embedding operations, and that recent theoretical results on the universality of GNNs [6], [7] can be exploited to define universal approximators for any pooling operator with continuous SRC functions.

4) We identify three evaluation criteria that allow quantifying how much a pooling operator preserves information related to the node attributes, how much it preserves the...
original topological structure, and how well it can adapt the coarsened graph to the downstream tasks. Using these criteria, we evaluate the performance and analyze the behavior of different classes of pooling operators. We provide guidelines to identify the appropriate pooling methods for a given task based on the taxonomy and the fulfillment of the criteria.

5) Finally, we release an open-source Python application programming interface (API) for implementing graph pooling operators. Such general interfaces are the basis on which popular software libraries for creating GNNs are built, as they offer an easy and flexible way to make novel operators available to the research community and to ensure reproducibility.

In Section II, we review the pooling operators present in the literature. Then, we introduce the SRC framework (Section III), present a taxonomy based on SRC (Section IV), and show how it allows us to categorize pooling operators in an orderly and user-friendly manner. We then present strategies to evaluate pooling methods and contrast their strengths and weaknesses (Section V). Finally, we provide guidelines to select appropriate pooling operators for the task at hand.

II. POOLING IN GNNs

In this section, we provide a brief overview of many pooling operators proposed in recent literature. We will present an organized taxonomy of these methods in Section IV.

In some of the seminal works on GNNs, the Graclus algorithm is used by [11] and later works on GNNs [12]–[14]. Graclus halves the node set by iteratively collapsing a randomly selected node with its most strongly connected neighbor. The method is equivalent to more expensive approaches based on the eigendecomposition of the adjacency matrix.

The literature about machine learning for point clouds (a field of computer vision that models 3-D objects as a collection of points in space) also introduces pooling techniques to generalize the traditional pooling layers of CNNs. Simonovsky and Komodakis [15] adopt the VoxelGrid algorithm, in which a regular grid is overlaid on a point cloud, and all the points in a voxel are summarized by their centroid. A similar voxel-oriented pooling is also proposed by [16]. Another approach for pooling point clouds is proposed by [17], where a hierarchical representation is obtained by clustering points around a given set of centroids, found with a farthest point sampling algorithm. A recent work by [18] proposes a pooling strategy based on octree partitioning [19].

Other pooling techniques are based on different design principles and requirements. For instance, the graph-theoretical analyses proposed by [20]–[22] focus on coarsening graphs, so that their spectral structure is preserved. In clique pooling [23], graphs are coarsened by aggregating maximal cliques. All nodes belonging to a clique are summarized by their maximum or average values and become a new node in the coarsened graph. If a node shares an edge with a node from another clique, the nodes representing the two cliques are connected in the new graph. Bacciu and Di Sotto [24] develop a pooling method based on the non-negative matrix factorization (NMF) of the adjacency matrix. After the decomposition, one of the two factors is used as a soft clustering matrix to coarsen the graph. In the EigenPooling approach by [25], a graph is first partitioned into subgraphs using spectral clustering, and each subgraph is mapped to a node in the pooled graph. Then, the eigenvectors of each subgraph’s Laplacian are used to define a set of downsampling operators for the node attributes. In Laplacian pooling (LaPool) [5], nodes characterized by high local quadratic variation among their features and those of their neighbors are selected as leaders, and the remaining nodes are assigned to one cluster using a sparse attention mechanism. When combined with simple low-pass graph filters, such as graph convolutional network (GCN), LaPool yields a bandpass filter that retains the medium frequencies of the graph signal. Xie et al. [26] propose a procedure that iteratively collapses nodes with high similarity and similar neighborhoods. This approach is similar to Graclus, although it accounts for the second-order similarity relations between nodes. Node Decimation Pooling (NDP) [27] partitions the nodes into two maximally similar groups based on the maximum cut objective. One group is dropped, while the other is kept, and a new graph is built by connecting the retained nodes through node neighborhood.

The current trend (and state of the art) in graph pooling points toward learnable operators that, much like convolutional layers, can dynamically adapt to a particular task to compute optimal pooling. The DiffPool operator [29] is among the first attempts to learn a pooling operator end-to-end. In DiffPool, a GNN is trained to compute a soft clustering matrix from the node features, which is then used to aggregate the nodes in each cluster. To obtain balanced clusters, two additional loss terms are minimized during training, namely, a link prediction loss and an entropy loss. In more recent work, [30] propose the Mapper-based PageRank (MPR), a pooling method based on the Mapper algorithm using PageRank as a lens function. Other approaches for computing learnable lens functions can be used; the authors also show that their Deep Graph Mapper algorithm is a generalization of those pooling methods based on a soft clustering, such as DiffPool and LaPool. Inspired by spectral clustering, MinCut pooling learns a soft cluster assignment matrix from the node features and the graph connectivity by optimizing a differentiable normalized cut objective.

Despite being very effective in practice, methods based on learning a clustering operation have been criticized for their high memory cost. To address these limitations, several works have proposed a family of sparse operators collectively known as Top-$K$ methods. In these approaches, node features are projected to a scoring vector through a learnable transformation, and the scoring vector is then used to decide which nodes to keep in the coarsened graph. In their simplest formulation, Top-$K$ approaches learn a linear projection to compute the scoring vector. Then, the highest $K$ elements of the scoring vector identify $K$ nodes to keep in the coarsened graph. An alternative strategy is proposed by [3], where all nodes with a score above a given threshold are kept. The threshold can be either selected man-

---

1https://github.com/danieleg rattarola/SRC

---

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
**TABLE I**

**POOLING METHODS IN THE SRC FRAMEWORK AND HOW THEY FIT INTO THE TAXONOMY. METHODS ARE DIVIDED INTO TRAINABLE OR NON-TRAINABLE (T/nT), DENSE OR SPARSE (D/S), AND FIXED OR ADAPTIVE (F/A). GNN INDICATES A STACK OF ONE OR MORE MESSAGE-PASSING LAYERS, MLP IS A MULTILAYER PERCEPTRON, L IS THE NORMALIZED LAPLACIAN, \( \beta \) IS A REGULARIZATION VECTOR (SEE [5]), D IS THE DEGREE MATRIX, \( \mu_{\text{max}} \) IS THE EIGENVECTOR OF THE LAPLACIAN ASSOCIATED WITH THE LARGEST EIGENVALUE, \( \mathbf{I} \) IS A VECTOR OF INDICES, AND \( A_1 \) SELECTS THE ROWS AND COLUMNS OF \( A \) ACCORDING TO \( \mathbf{i} \).**

| Method   | Select | Reduce | Connect | T | nT | D | S | F | A |
|----------|--------|--------|---------|---|----|---|---|---|---|
| DiffPool | \( S = \text{GNN}_1(\mathbf{A}, \mathbf{X}) \) (w/ auxiliary loss) | \( \mathbf{X}' = \mathbf{S}^T \cdot \text{GNN}_2(\mathbf{A}, \mathbf{X}) \) | \( \mathbf{A}' = \mathbf{S}^T \mathbf{A} \) | ✓ | ✓ | ✓ | ✓ | ✓ |
| MinCut   | \( S = \text{MLP}(\mathbf{X}) \) (w/ auxiliary loss) | \( \mathbf{X}' = \mathbf{S}^T \mathbf{X} \) | \( \mathbf{A}' = \mathbf{S}^T \mathbf{A} \) | ✓ | ✓ | ✓ | ✓ | ✓ |
| Top-K    | \( y = \text{XP}(\mathbf{p}) \); \( i = \text{topK}(y) \) | \( \mathbf{X}' = (\mathbf{X} \odot \sigma(y))_{i} \); \( \mathbf{A}' = \mathbf{A}_{l,i} \) | ✓ | ✓ | ✓ | ✓ | ✓ |
| SAGPool  | \( y = \text{GNN}(\mathbf{A}, \mathbf{X}); i = \text{topK}(y) \) | \( \mathbf{X}' = (\mathbf{X} \odot \sigma(y))_{i} \); \( \mathbf{A}' = \mathbf{A}_{l,i} \) | ✓ | ✓ | ✓ | ✓ | ✓ |
| NMF      | \( \text{Factorize}: \mathbf{A} = \mathbf{WH} \rightarrow \mathbf{S} = \mathbf{H}^T \) | \( \mathbf{X}' = \mathbf{S}^T \mathbf{X} \) | \( \mathbf{A}' = \mathbf{S}^T \mathbf{A} \) | ✓ | ✓ | ✓ | ✓ | ✓ |
| LaPool   | \( \{ i \mid (i,j) \in \mathcal{N}(i): V_{i} > V_{j} \} \) | \( \mathbf{X}' = \mathbf{S}^T \mathbf{X} \) | \( \mathbf{A}' = \mathbf{S}^T \mathbf{A} \) | ✓ | ✓ | ✓ | ✓ | ✓ |
| Graclus  | \( S_k = \{ (\mathbf{x}_i, \mathbf{x}_j) \mid \arg \max_{\mathbf{x}} (\mathbf{A}_{ij} D_{ii}^{-1/2} + \mathbf{A}_{ij} D_{jj}^{-1/2}) \} \) | \( \mathbf{X}' = \mathbf{S}^T \mathbf{X} \) | METIS | ✓ | ✓ | ✓ | ✓ | ✓ |
| NDP      | \( i = \{ i \mid u_{\text{max}, i} > 0 \} \) | \( \mathbf{X}' = \mathbf{X}_i \) | Kron red. | ✓ | ✓ | ✓ | ✓ | ✓ |

IIII. SELECT, REDUCE, AND CONNECT

**A. Notation**

We use symbols \( \mathbb{N} \) and \( \mathbb{R} \) for the sets of natural and real numbers, and calligraphic letters for generic sets. Integer numbers are denoted with uppercase letters, e.g., \( N, F \in \mathbb{N} \), while indices are in lowercase letters. Feature vectors and matrices are in bold lowercase and uppercase letters, respectively, e.g., \( \mathbf{x} \in \mathbb{R}^N \) and \( \mathbf{X} \in \mathbb{R}^{N 	imes F} \). We denote a graph with \( N \) nodes as a tuple \( \mathcal{G} = (\mathcal{X}, \mathcal{E}) \), where \( \mathcal{X} = \{1, 2, \ldots, N\} \) is the set of nodes, and \( \mathcal{E} \subseteq \mathcal{X} \times \mathcal{X} \) is the set of edges. When a graph \( \mathcal{G} \) is attributed, its nodes and edges are associated with attributes or features. Accordingly, we denote by \( \mathcal{X} = \{ (i, \mathbf{x}_i) \}_{i=1:N} \) the set of nodes \( i \) with attribute \( \mathbf{x}_i \), in some space \( \mathcal{X} \); similarly, \( \mathcal{E} = \{ (i, j), e_{ij} \}_{i,j=1:N} \) is the edge set, where \( e_{ij} \in \mathcal{E} \) is the attribute from some space \( \mathcal{E} \) that is associated with the edge from node \( i \) to node \( j \). Usually, \( \mathcal{X} \) and \( \mathcal{E} \) are real vector spaces, namely, \( \mathcal{X} = \mathbb{R}^F \) and \( \mathcal{E} = \mathbb{R}^H \) for \( F, H \in \mathbb{N} \). When there is no ambiguity, in the following, we simplify the notation by identifying node \( (i, \mathbf{x}_i) \) with its attribute \( \mathbf{x}_i \) only, and edge \( [i, j], e_{ij} \) with \( e_{ij} \). In most cases, it is also practical to represent the graph with an adjacency matrix \( \mathbf{A} \in \{0, 1\}^{N \times N} \) and a node attribute matrix \( \mathbf{X} \in \mathbb{R}^{N \times F} \). In this article, we consider undirected graphs, since the literature on graph pooling mostly focuses on them.

Let a graph pooling operator be loosely defined as any function \( \text{POOL} \) that maps a graph \( \mathcal{G} \) to a new pooled graph \( \mathcal{G}' = (\mathcal{X}', \mathcal{E}') \), with the generic goal of reducing the number of nodes from \( N \) to \( K < N \). To facilitate our study of graph pooling methods, it is useful to isolate the main operations that all methods must perform, regardless of their specific implementation. We identify three operations: SRC; see Fig. 1. The selection operation computes \( K \) subsets of nodes, each associated with one node of the output \( \mathcal{G}' \); we refer to them as supernodes. The reduction operation aggregates the node attributes from each supernode to obtain the node attributes of \( \mathcal{G}' \). Finally, the connection operation computes edges among the \( K \) reduced nodes.

The SRC operations allow us to easily describe pooling methods, as done in Table I. Accordingly, we define a pooling operator as any function \( \text{POOL} : \mathcal{G} \mapsto \mathcal{G}' = (\mathcal{X}', \mathcal{E}') \) written as the composition of

\[
S = \text{SEL}(\mathcal{G}) = \{ S_k \}_{k=1:K}; \quad \text{(Selection)}
\]

\[
\mathcal{X}' = \{ \text{RED}(\mathcal{G}, S_k) \}_{k=1:K}; \quad \text{(Reduction)}
\]

\[
\mathcal{E}' = \{ \text{CON}(\mathcal{G}, S_k, S_l) \}_{k,l=1:K}; \quad \text{(Connection)}
\]

Different pooling operators are determined by the specific implementations of \( \text{SEL}, \text{RED}, \text{and CON} \).

**B. Select**

The selection function \( \text{SEL} \) maps the nodes of the input graph to the nodes of the pooled one. The role of \( \text{SEL} \) is crucial, as it determines the number of nodes in the output

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
C. Reduce

The reduction function computes the node attributes of graph $G'$ by aggregating the node attributes of $G$ associated with each supernode $S_k$. A reduction consists of applying a function RED to each supernode $S_k$ to produce the $k$th node attribute $x_k'$ of $G'$: $\text{RED} : G', S_k \mapsto x_k' \in \mathcal{X}$.

D. Connect

The connection function determines, for each pair of supernodes $S_k, S_l$, the presence or absence of an edge between the corresponding nodes $k$ and $l$ in the pooled graph. The function also computes the attributes to be assigned to new edges and reads: $\text{CON} : G', S_k, S_l \mapsto e_{kl} \in \mathcal{E}$. We assume that the space of edge attributes contains a null attribute encoding the absence of an edge, and that the edge set of a graph only contains non-null edges.

The reason why both RED and CON are defined as the functions of graph $G'$ is that their output can depend on the full topology of the input graph in non-trivial ways. For example, pooling methods based on the graph spectrum often connect the nodes of $G'$ based on the whole structure of $G$ [27]. However, we notice that many pooling operators implement RED and CON as the functions of the supernodes only.

Given our definition of pooling operators as a combination of the SRC functions, we show in Table I how to express several pooling methods proposed in recent literature under the SRC formalism. We observe that SEL is commonly expressed as a matrix $S \in \mathbb{R}^{N \times K}$, where $S_{ik}$ indicates the membership score of node $i$ to supernode $k$, and $S_{ik} = 0$ means that node $i$ is not assigned to supernode $k$.

### IV. Taxonomy of Graph Pooling

The SRC framework is a general template to describe pooling operators, and it allows us to characterize the different families of pooling methods found in the literature. We propose the following taxonomy of pooling operators based on four distinguishing characteristics. We show in Tables I and II how different pooling methods fit into this taxonomy.

#### A. Trainability

The first distinction among pooling operators is whether $\text{SEL}$, $\text{RED}$, and $\text{CON}$ are learned end-to-end as part of the overall GNN architecture. In this case, we say that a method is trainable, i.e., the operator has parameters that are learned by optimizing a task-driven loss function, while in all other cases, we say that the methods are non-trainable. This distinction is important, because while non-trainable methods are often used as stand-alone algorithms for graph coarsening, trainable methods were specifically designed for GNNs and are a novel research topic of their own.

In general, non-trainable methods are useful when there is strong prior information about the desired behavior of pooling (e.g., preserving connectivity [10] or filtering out some particular graph frequencies [5]). These prior assumptions are usually grounded on graph-theoretical properties and are useful when few data are available, since they do not increase the overall number of parameters and introduce no additional optimization objectives when training the GNN. A well-known example of non-trainable pooling is the conventional grid pooling of CNNs, which pools spatially localized groups of pixels. On the other hand, trainable methods are more flexible and make fewer assumptions about the desired result. Therefore, they are useful in problems where the best pooling strategy is not known a priori. However, note that it is possible to integrate priors about the desired pooling behavior also in trainable methods (e.g., the MinCut operator also optimizes a normalized cut objective to ensure that supernodes correspond to well-separated node clusters of similar size). These additional assumptions usually act as a regularization in the GNN optimization.

#### B. Density of the Supernodes

A second axis of the taxonomy is concerned with the size of supernodes and the consequent cost of computing the selection function. We define the density of a pooling operator

---

**TABLE II**

**Classification of Pooling Operators According to the Proposed Taxonomy**

| Method          | T | nT | D | S | F | A | H | G |
|-----------------|---|----|---|---|---|---|---|---|
| DiffPool [29], MinCut [33], StructPool [40] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Top-K methods [35, 34, 36, 3, 37, 41], Edge Contract. [39] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Coates and Ng [42], Voxelization-based [15, 16, 17, 18] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NMF [24], EigenPooling [25], LaPool [5], Clique [23] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Xie et al. [26], MPR [30] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Gracuis [10], NDP [27], Pooling in CNNs | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [43, 44, 45] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| [46, 47, 48, 49, 50] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SortPool [51] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Scarselli et al. [52] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
as the expected value $\mathbb{E}[|S_i|/N]$ of the ratio between the cardinality of a supernode $S_i$ and the number of nodes in $\mathcal{G}$. We say that a method is dense if SEL generates supernodes $S_i$ whose cardinality is $O(N)$, and sparse if supernodes have constant cardinality $O(1)$.² Fig. 2(a) shows an example of sparse and dense selection.

This distinction is key, since sparse methods require much less computational resources, especially in terms of memory, which is a significant bottleneck even in modern GPUs. This makes them scale better to large graphs. However, as we show in Section V, sparse selection is a harder operation to learn than dense selection and may result in unexpected behaviors.

C. Adaptability of $K$

It is also possible to distinguish pooling methods according to the number of nodes $K$ of the pooled graph. If $K$ is constant and independent of the input graph size, we say that a pooling method is fixed. In this case, $K$ is a hyperparameter of the pooling operator, and the output graph will always have $K$ nodes. For example, $K$ can be the number of output features of a neural network used to compute cluster assignments [29], [33]. On the other hand, if the number of supernodes is a function $K(\mathcal{G})$ of the input graph, we say that the method is adaptive. In many cases, $K(\mathcal{G})$ is a function of $N$ (e.g., the ratio $N/2$), but $K(\mathcal{G})$ could also depend on the input graph in a more complex way (e.g., [3], [5]).

Adaptive pooling methods can compute graphs that have a size proportional to that of the input. On the other hand, all the coarsened graphs generated by fixed methods will have the same size. This can lead to situations where $K > N$ for some graphs, causing them to be upscaled by pooling, rather than coarsened. Fig. 2(b) compares fixed and adaptive pooling and shows an example (second row) where fixed pooling upscales the graph. For data with a wide or skewed distribution of the number of nodes, the values commonly chosen for the graph. For data with a wide or skewed distribution of the graph. For data with a wide or skewed distribution of the graph. For data with a wide or skewed distribution of the graph. For data with a wide or skewed distribution of the graph. For data with a wide or skewed distribution of the graph.

D. Hierarchy

A distinction often found in the literature is that between “regular” and global pooling, which is extremely evident, to the point where global pooling is usually referred to as a separate operation called “readout.” Here, we show that such a distinction can be formalized with the SRC framework. In particular, global pooling indicates those methods that reduce a graph to a single node, discarding all topological information. A pooling method is global if it is fixed with $K = 1$, i.e., it returns a degenerate single-node graph represented by its attribute. Also, the connection function is a constant map to the empty set. On the other hand, we indicate all other methods as hierarchical pooling operators. Fig. 2(a) shows an example of hierarchical and global pooling.

Hierarchical and global pooling operators have different roles, and both can be part of the same GNN architecture for graph-level learning. The former provide a multiresolution representation of the graph from which the GNN can gradually distill high-level properties, while the latter compute graph embeddings to interface with traditional layers operating on vectors.

E. Discussion

We observe that the following hold.

1) The main differences among pooling methods are in the selection function, while much less variety is found in the reduction and connection functions.

2) A majority of methods (e.g., [5], [15], [17], [23]–[25]) have adaptive $K$, with a dense and non-trainable selection.

3) Adaptive methods are the most commonly found in the literature, although fixed pooling operators are currently the state of the art [29], [33].

4) We also note that, to the best of our knowledge, there are no pooling (No-pool) operators that are trainable, dense, and adaptive, which could be an interesting research topic in the near future.

Considering density and adaptability, we see that the memory cost of a pooling operator can range from $O(1)$ (sparse and fixed) to $O(N^2)$ (dense and adaptive). This is especially relevant for trainable methods, which usually need to fit into memory-bound computational units, such as GPUs and tensor processing units (TPUs). In this regard, we empirically observed that sparse trainable methods can pool graphs up to four times bigger than dense ones when using a typical GPU (details in the Supplementary Material).

F. SRC as Embedding Operations

Observing that each of the three SRC functions is, essentially, either a graph-level or a node-level embedding, we are able to analyze the expressive power of different pooling operators using recent results developed for generic GNNs [53]. By representing the output of the selection function as a membership score matrix $S \in \mathbb{R}^{N \times K}$, function SEL can be seen as a node-level embedding function where each node $i$ in $\mathcal{G}$ is mapped to a feature vector $S_{ni} \in \mathbb{R}^K$ representing the membership of node $i$ to each of the $K$ supernodes of $\mathcal{G}' = \text{POOL}(\mathcal{G})$. Now, let $\mathcal{G}'$ indicate an augmentation of $\mathcal{G}$,
such that its node features are defined as \( X_{S_k} = X||S_{S_k} \), where \( \| \) indicates concatenation; similarly, let \( G(S_k, S_l) \) be defined by \( X(S_k, S_l) = X || S_{S_k} || S_{S_l} \). It is easy to see that functions \( \text{RED} \) and \( \text{CON} \) are graph-level embedding operations of the form

\[
\text{RED} : G_{S_k} \mapsto x'_k; \quad \text{CON} : G(S_k, S_l) \mapsto e'_{kl}.
\]

Typically, we request that \( \text{SEL} \) is equivariant under the permutation of the nodes of \( G \) and that \( \text{RED} \) and \( \text{CON} \) are invariant to node permutations,\(^4\) so that they are consistent with respect to different representations (i.e., permutations of the nodes) of the same graph.

An interesting consequence of this interpretation of SRC is that graph pooling operators inherit the results about the expressivity of GNNs. For example, we can design a universal pooling operator that approximates any arbitrary function to generate a coarsened graph by implementing \( \text{SEL} \) as a universal equivariant network \([6]\), and \( \text{RED} \) and \( \text{CON} \) as universal invariant networks \([7]\).

V. Evaluation

We argue that there is no single general-purpose measure to quantify the performance of a graph pooling algorithm and the quality of a coarsened graph. In this section, we define three evaluation criteria for pooling operators and design experiments to test whether different classes of methods are able to meet them. In particular, we evaluate operators based on their ability to: 1) preserve the information content of the node attributes; 2) preserve the topological structure; and 3) preserve the information required to solve various classification tasks. The goal here is to contrast the categories of the proposed taxonomy according to these three criteria. To perform the comparison, we consider the eight hierarchical pooling methods of Table I as representatives: MinCut \([33]\) and DiffPool \([29]\) are trainable, dense, and fixed; Top-K \([34], [35]\) and SAGPool \([36]\) are trainable, sparse, and adaptive; NMF \([24]\) and LaPool \([5]\) are non-trainable, dense, and adaptive; Graclus \([10]\) and NDP \([27]\) are non-trainable, sparse, and adaptive. All implementation details are in the Supplementary Material.

A. Preserving Node Attributes

As a first experiment, we test the ability of pooling methods to preserve node information. We consider the task of reconstructing the original coordinates of a geometric point cloud from its pooled version. We configure a graph autoencoder to pool the node attributes and then lift them back to the original size using an appropriate lift operator for each method (details in the Supplementary Material). Note that this experiment evaluates the quality of the pooling methods in compressing node information, but it does not test their generalization capability, since the autoencoder is independently fit on each point cloud.

Table III reports the average and standard deviation of the mean squared error (MSE) obtained by the eight methods on different point clouds from the PyGSP library \([54]\) and the ModelNet40 dataset \([55]\).

As baseline values for the MSE, we report the mean squared distance between features of adjacent vertices: \( \gamma = (\langle |F|^2 \rangle)^{-1} \sum_{(i,j) \in E} |x_i - x_j|^2 \); we recall from Section III that \( E \) is the edge set, \( x_i \in \mathbb{R}^2 \) is the feature vector of node \( i \) (coordinates of \( i \)th point), and \( |\cdot|^2 \) is the squared Euclidean norm. This reference score tells us whether the reconstructed nodes, on average, are close to their target. By “close,” we mean that the MSE is less than a reference value computed as the average distance between the neighbors of the target graph, i.e., the average length of the blue segments in the inset figure. In the figure, we also see green segments that are shorter than the reference value, while red ones are longer. In other words, for this problem, we may consider a reconstruction to be successful if the average of red and green segments is lower than the average of the blue segments.

We observe that, as the point clouds grow in size, many operators cannot achieve MSE < \( \gamma \) (red entries). The two methods that stand out from Table III are the non-trainable

---

4Equivariance to node permutations implies that reordering the nodes in the input graph results in output vectors (node memberships) permuted accordingly.

5Node-permutation invariance implies that the output is the same regardless on the ordering of the input nodes.

6“OOR” indicates out of resources, i.e., either we could not fit a batch size of eight graphs on an Nvidia Titan V GPU or it took more than 24 h to complete training. The values of 0.000 indicate any value <10\(^{-6}\).
NMF and NDP, as confirmed also by their respective average ranks across datasets. We include figures of the original and reconstructed graphs for Table III, Supplementary Material.

**Interpreting the Selection Operation:** Fig. 3 depicts a variant of the coarsened graphs where the SEL and CON operations are the same as in Table I, but the RED function is replaced by

\[ X' = S^\top X. \]

This modification is crucial to interpret the SEL operation, as most of the pooling methods use message passing layers before the reduction (see the autoencoder architecture details), which makes the node feature space \( X' \) not directly comparable with the original 2-D or 3-D input space. Conversely, the reduction in (2) gives (weighted) averages of the supernodes with the benefits of maintaining points in the input space and locating them in the supernodes’ centers of mass.

Two main patterns emerge. First, we see that the two non-trainable sparse methods (NDP and Graculus) perform a rather uniform node subsampling, in such a way that the reduced node features are a good representative of the original input, which may facilitate the reconstruction of the input node features, as confirmed by the low MSE in Table III. Second, trainable and sparse methods (TopK and SAGPool) tend to cut off entire portions of the graphs, therefore discarding essential node information.

**B. Preserving Structure**

In this experiment, we study the structural similarity between the input and coarsened graphs \( G \) and \( G' \), respectively, by comparing the quadratic forms associated with their respective combinatorial Laplacian matrices \( L \) and \( L' \). This evaluation criterion has also been recently studied by [20]–[22] and allows us to compare graphs of different sizes. In particular, we consider the quadratic loss 

\[ L(G, G') = \sum_{i,j} (L_{ij} - L'_{ij})^2, \]

where \( X \) is an arbitrary graph signal, and \( X' \) its reduction. In this experiment, we choose \( X \) to be the concatenation of the first ten eigenvectors of \( L \) and the node coordinates of \( G \); all columns are 2-normalized. For trainable methods, we directly minimize the loss as a self-supervised target. Table IV reports the average loss obtained by the eight operators on different graphs from the PyGSP library and also on three popular citation networks (Cora, Citeseer, and Pubmed), while Fig. 4 shows the examples of pooled graphs and their spectra. We show the result for Grid2d, since it is easier to interpret visually; figures for all graphs are available in the Supplementary Material.

Trainable dense methods can generate coarsened graphs with a quadratic loss with respect to the original graph lower than their non-trainable or sparse counterparts. Interestingly, from the bottom row of Fig. 4, we see that a low quadratic loss does not necessarily imply a good alignment of the spectra. For example, on the regular grid in Fig. 4, the excellent spectral alignment achieved by Top-K and SAGPool is not reflected by a low quadratic loss value (0.596 and 0.361, respectively). While, in principle, this experiment focuses on comparing only SEL and CON, we are also evaluating RED, since it affects the loss that depends on \( X' \). This can explain the discrepancy between the loss values and the eigenvalue plots.
Properties of the Connection Operation: From Fig. 4, we see that dense methods (DiffPool, MinCut, NMF, and LaPool) yield coarsened graphs that are densely connected. We can make a similar observation for the autoencoder experiment with modified RED operation in (2), as shown in Fig. 3. However, in these dense graphs, most of the edge weights are also small. This is quantitatively reported in Table V, in which we compare the density of edges ($|E|/K^2$) and the median edge weight of the coarsened graphs for Grid2d, Minnesota, and Sensor. An extended version of Table V is reported in the Supplementary Material.

C. Preserving Task-Specific Information

In our final experiment, we consider several benchmarks of graph classification to test the third criterion. A high classification accuracy implies that an operator can selectively preserve information based on the requirements of the task at hand. We consider graph classification problems from the TUDataset [56], the ModelNet10 dataset [55], and the Colors-3 and Triangles datasets introduced by [3]. Table VI reports the average and standard deviation of the classification accuracy on the test set, as well as the average ranking of the operators. We also report as a baseline the classification accuracy of a GNN with No-pool. We observe that, on the datasets considered here, the operators based on graph spectral properties (MinCut, NDP, and Graclus) achieve the highest accuracy. However, we could not find strong evidence that one pooling operator (or even a class of operators) is systematically better than all others. For instance, on Triangles and Colors-3, we see that dense,
trainable operators have a consistent advantage. However, the family of sparse and/or non-trainable methods achieves a better performance on Enzymes, Mutagenicity, and the large-scale ModelNet10 datasets. Finally, in datasets, such as Mutagenicity, Proteins, and DD, the performance gap is not very large. Table VI also shows that some of the models with graph pooling operators achieve higher classification accuracy than the No-pool baseline (in green). In Mutagenicity, the baseline architecture achieves top performance, suggesting that graph pooling is not always beneficial in certain graph classification tasks. Further discussion can be found in the recent work of [4].

Preserving task-specific information is a criterion that can also be used to evaluate global pooling methods. We run the same graph classification benchmarks using a GNN architecture where we only replace the final global pooling layer. We consider three of the most common techniques for global pooling, namely, summing, averaging, and taking the element-wise max of the node features, and also two methods from the literature: the attention-based pooling proposed by [43] (trainable, dense) and SortPool [51] (non-trainable, sparse). The results, reported in Table VII, show that the choice of global pooling method can have substantial impact on the performance of a GNN architecture, with the trainable attention-based method outperforming the others (perhaps unsurprisingly, given the additional parameters) and the maximum pooling leading among the non-trainable ones.

VI. CONCLUDING REMARKS AND GUIDELINES

In this article, we presented SRC, a unifying formulation of pooling operators in GNNs, that allowed us to organize the vast literature on the subject under a comprehensive taxonomy and implement every pooling operator under a well-defined and modular framework.

Overall, we showed that the choice of the best pooling operator, and whether performing graph pooling is necessary at all, highly depends on the problem at hand. A comprehensive evaluation of pooling operators requires considering multiple criteria to highlight all their fundamental properties, and, as such, it cannot be limited to measuring the downstream performance on a few small-scale benchmark datasets.

A. Guidelines

For the abovementioned reasons, we provide guidelines to choose a pooling method in practice, based on the three proposed evaluation criteria and the taxonomy.

1) To preserve node attributes, especially in point clouds, non-trainable and sparse methods are suggested, since they usually compute a uniform coarsening of the graph.
2) To preserve structure, dense and trainable methods are better at minimizing the Laplacian quadratic loss, although sparse methods yield a better spectral alignment.
3) To preserve task-specific information, which is the most common setting in machine learning, trainable methods have an advantage over their counterpart, although we conclude that there is no better method a priori. Non-trainable sparse methods (Graculus and NDP) have overall better performance across different tasks and are advised as the first choice; however, if the goal is to optimize for a specific objective, then trainable dense methods (MinCut and DiffPool) offer more flexibility and are more easily integrated into GNN architectures.
4) We also observe that trainable sparse methods discard entire portions of the graphs, and therefore, they are generally less advisable.
5) For global pooling, the trainable method that we tested achieved a consistently better performance than the other baselines, which confirms our observation regarding trainable methods. Maximum pooling appears to be a good choice among non-trainable methods.

We believe that SRC will be helpful in further studying graph pooling, and that our analysis will guide practitioners in choosing an appropriate pooling method for the application at hand. Our work also leads to a more principled approach in designing and evaluating new pooling operators and will help the research community to advance the field.

ACKNOWLEDGMENT

The authors would like to thank Nvidia Corporation, Santa Clara, CA, USA, for their support with the donation of the Titan XP GPU used for this work.

REFERENCES

[1] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” 2017, arXiv:1704.01212.
[2] P. W. Battaglia et al., “Relational inductive biases, deep learning, and graph networks,” 2018, arXiv:1806.01261.
[3] B. Knyazev, G. W. Taylor, and M. R. Amer, “Understanding attention and generalization in graph neural networks,” 2019, arXiv:1905.02850.
[4] D. Mesquita, A. Souza, and S. Kaski, “Rethinking pooling in graph neural networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 33, 2020, pp. 1–12.
[5] E. Noutahi, D. Beaini, J. Horwood, S. Giguère, and P. Tossou, “Towards interpretable sparse graph representation learning with Laplacian pooling,” 2019, arXiv:1905.11577.
[6] N. Keriven and G. Peyré, “Universal invariant and equivariant graph neural networks,” in Proc. Adv. Neural Inf. Process. Syst. Red Hook, NY, USA: Curran Associates, 2019, pp. 7090–7099.
[7] H. Maron, E. Fetaya, N. Segol, and Y. Lipman, “On the universal- ity of invariant networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 4363–4371.
[8] M. Fey and J. Eric Lenssen, “Fast graph representation learning with PyTorch geometric,” 2019, arXiv:1903.02428.
[9] D. Grattarola and C. Alippi, “Graph neural networks in TensorFlow and Keras with Spektral,” IEEE Comput. Intell. Mag., vol. 16, no. 1, pp. 99–106, Feb. 2021.
[10] I. S. Dhillon, Y. Guan, and B. Kulis, “Weighted graph cuts without eigenvectors a multilevel approach,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 11, pp. 1944–1957, Nov. 2007.

[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] 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), vol. 1, Jul. 2017, pp. 5115–5124.

[13] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, “CayleyNets: Graph convolutional neural networks with complex rational spectral filters,” 2017, arXiv:1705.07664.

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

[15] 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.

[16] G. Riegler, A. O. Ulusoy, and A. Geiger, “OctNet: Learning 3D representations at high resolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 9631–9640.

[17] D. Meagher, “Geometric modeling using octree encoding,” Comput. Graph. Image Process., vol. 19, no. 2, pp. 129–147, Jun. 1982.

[18] H. Lei, N. Akhtar, and A. Mian, “Octree guided CNN with spherical kernels for 3D point clouds,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 9606–9615.

[19] A. Loukas, “Graph reduction with spectral and cut guarantees,” in Proc. ICLR, 2020, pp. 1–8.

[20] E. Luzhnica, B. Day, and P. Liò, “Clique pooling for graph classification,” in Proc. Conf. Int. Conf. Learn. Represent. (ICLR), 2019, pp. 1–9.

[21] 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.

[22] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, “Graph convolutional networks with EigenPooling,” 2019, arXiv:1904.13107.

[23] Y. Xie, C. Yao, M. Gong, C. Chen, and A. K. Qin, “Graph convolutional networks with multi-level cross-modal coarsening for graph classification,” Knowl.-Based Syst., vol. 194, Apr. 2020, Art. no. 105578.

[24] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, “Hierarchical representation learning in graph neural networks with node decimation pooling,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 5, pp. 2195–2207, May 2022.

[25] F. Dörfler 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. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” 2018, arXiv:1806.08804.

[27] C. Bodnar, C. Cangea, and P. Liò, “Deep graph mapper: Seeing graphs through the neural lens,” 2020, arXiv:2002.03864.

[28] G. Singh, F. Ménoli, and G. E. Carlsson, “Topological methods for the analysis of high dimensional data sets and 3D object recognition,” in Proc. SPDEP, 2007, pp. 91–100.

[29] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Stanford InfoLab, Stanford, CA, USA, Tech. Rep. 422, 1999.

[30] F. M. Bianchi, D. Grattarola, and C. Alippi, “Spectral clustering with graph neural networks for graph pooling,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 874–883.

[31] C. Cangea, P. Velicković, N. Jovanović, T. Kipf, and P. Liò, “Towards sparse hierarchical graph classifiers,” 2018, arXiv:1811.01287.

[32] H. Gao and S. Ji, “Graph U-Net,” in Proc. ICLR, 2019, pp. 1–10.

[33] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling,” 2019, arXiv:1904.08082.

[34] Z. Zhang et al., “Hierarchical multi-view graph pooling with structure learning,” IEEE Trans. Knowl. Data Eng., early access, Jun. 18, 2021, doi: 10.1109/TKDE.2021.3090664.

[35] A. Coates and A. Y. Ng, “Selecting receptive fields in deep networks,” in Proc. Adv. Neural Inf. Process. Syst., 2011, pp. 2528–2536.

[36] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” 2015, arXiv:1511.05493.

[37] F. Diehl, T. Brunner, M. T. Le, and A. Knoll, “Towards graph pooling by edge contraction,” in Proc. ICML Workshop Learn. Reasoning Graph-Based Represent. Structured, 2019, pp. 1–8.

[38] F. Diehl, “Edge contraction pooling for graph neural networks,” 2019, arXiv:1905.10990.

[39] H. Yuan and S. Ji, “StructPool: Structured graph pooling via conditional random fields,” in Proc. Int. Conf. Learn. Represent., 2020, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=BJxhVwH

[40] E. Ranjan, S. Sanyal, and P. P. Talukdar, “ASAP: Adaptive structure aware pooling for learning hierarchical graph representations,” 2019, arXiv:1911.07979.

[41] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” 2015, arXiv:1511.06391.

[42] N. Navarin, D. V. Tran, and A. Sperduti, “Universal readout for graph convolutional neural networks,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2019, pp. 1–7.

[43] J. Wu, J. He, and J. Xu, “DEMO-Net: Degree-specific graph neural networks for node and graph classification,” 2019, arXiv:1906.02319.

[44] P. Corcoran, “Function space pooling for graph convolutional networks,” 2019, arXiv:1905.06259.

[45] J. Atwood and D. Towsley, “Diffusion-convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 1993–2001.

[46] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” 2018, arXiv:1810.00826.

[47] Y. Bai et al., “Unsupervised inductive graph-level representation learning via graph-graph proximity,” 2019, arXiv:1904.01098.

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

[49] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE Trans. Neural Netw., vol. 20, no. 1, pp. 61–80, Dec. 2009.

[50] R. Sato, “A survey on the expressive power of graph neural networks,” 2020, arXiv:2003.04078.

[51] M. Defferrard, L. Martin, R. Pena, and N. Perraudin. PyGSP: Graph Signal Processing in Python. Accessed: Apr. 8, 2022. [Online]. Available: https://github.com/epl-ils2/pygsp/

[52] Z. Wu et al., “3D ShapeNets: A deep representation for volumetric shapes,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 1912–1920.

[53] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, “TUDataset: A collection of benchmark datasets for learning with graphs,” 2020, arXiv:2007.08663.

Daniele Grattarola received the M.Sc. degree (Hons.) in computer science and engineering from the Politecnico di Milano, Milan, Italy, in 2017, and the Ph.D. degree from the Dalle Molle Institute for Artificial Intelligence (IDSIA), Università della Svizzera italiana, Lugano, Switzerland, in 2021. His research interests include graph neural networks and their applications to computational biology and dynamical systems.
Daniele Zambon (Member, IEEE) received the Ph.D. degree from the Università della Svizzera italiana (USI), Lugano, Switzerland, in 2021. He is currently a Post-Doctoral Researcher with the Swiss AI Laboratory, Dalle Molle Institute for Artificial Intelligence (IDSIA), Lugano, and the Università della Svizzera italiana (USI). He has been a Visiting Researcher/Intern with the University of Florida, Gainesville, FL, USA, the University of Exeter, Exeter, U.K., and STMicroelectronics, Geneva, Switzerland. His research interests include machine learning on graph-structured data, time series analysis, and learning in nonstationary environments.

Filippo Maria Bianchi is currently an Associate Professor with the Department of Mathematics and Statistics, UiT the Arctic University of Norway, Tromsø, Norway. He is also with the Norwegian Research Centre (NORCE), Bergen, Norway. His research interests include the intersection among machine learning, dynamical systems, and complex networks. The main areas where he applies his research are energy analytics and remote sensing.

Cesare Alippi (Fellow, IEEE) is currently a Professor with the Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland, and the Politecnico di Milano, Milan, Italy, a Visiting Professor with the University of Guangzhou, Guangzhou, China, and an Advisory Professor with the Northwestern Polytechnic, Xi’an, China. He was a Visiting Researcher/Professor with the University College London (UCL), London, U.K., Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, ESPCI Paris, Paris, France, CASIA, Trutnov, Czech Republic, Agency for Science, Technology and Research (A*STAR), Singapore, and UKobe, Kobe, Japan. He has authored or coauthored one monograph book, seven edited books, and about 200 papers in international journals and conference proceedings. He holds eight patents. His current research interests include adaptation and learning in non-stationary environments, graph learning and intelligence for embedded, the Internet of Things (IoT), and cyber-physical systems. Dr. Alippi is a fellow of the European Laboratory for Learning and Intelligent Systems, a member of the Administrative Committee of the IEEE Computational Intelligence Society (CIS), and a Board of Governors Member of the International Neural Network Society. He served the IEEE CIS in several capacities, also covering the roles of vice-president for education and chairing the awards committee. He received several awards including the International Neural Networks Society Gabor Award.