Graph Neural Networks with maximal independent set-based pooling: Mitigating over-smoothing and over-squashing

Stevan Stanovic, Benoit Gaüzère, Luc Brun

Graph Neural Networks with Maximal Independent Set-Based Pooling: Mitigating Over-smoothing and Over-squashing

Stevan Stanovic[1], Benoit Gaüzère[2], and Luc Brun[1]

[1]Normandie Univ, ENSICAEN, CNRS, UNICAEN, GREYC UMR 6072,14000 Caen, France
[2]INSA Rouen Normandie, Univ Rouen Normandie, Université Le Havre Normandie, Normandie Univ, LITIS UR 4108,F-76000 Rouen, France

Abstract

Graph Neural Networks (GNNs) have significantly advanced graph-level prediction tasks by utilizing efficient convolution and pooling techniques. However, traditional pooling methods in GNNs often fail to preserve key properties, leading to challenges such as graph disconnection, low decimation ratios, and substantial data loss. In this paper, we introduce three novel pooling methods based on Maximal Independent Sets (MIS) to address these issues. Additionally, we provide a theoretical and empirical study on the impact of these pooling methods on over-smoothing and over-squashing phenomena. Our experimental results not only confirm the effectiveness of using maximal independent sets to define pooling operations but also demonstrate their crucial role in mitigating over-smoothing and over-squashing.

1 Introduction

Graph Neural Networks (GNNs) (Scarselli et al. (2009)) are inspired by Convolutional Neural Networks (CNNs) and aim to apply representation learning to graphs. At the core of CNNs are convolution and pooling operations. Adapting these operations to graphs requires addressing unique challenges, such as the variable number of nodes, node order permutations, and structural irregularities.

In GNNs, convolution is primarily carried out through a message-passing scheme (MPNN) (Gilmer et al. (2017)), where the new representation of each node is computed by aggregating features from neighboring nodes in a permutation-invariant manner (Kipf & Welling (2017)). Similar to CNNs, pooling operations (Lee et al. (2019)) reduce the size of a graph or compress the entire graph into a vector (global pooling) after which a Multi-Layer Perceptron (MLP) makes the final decision.

Despite achieving state-of-the-art results on numerous graph prediction datasets, GNNs still suffer from phenomena like over-squashing and over-smoothing, which limit their predictive abilities, especially when layers are stacked. Over-smoothing (Rusch et al. (2023a)) occurs when node representations become increasingly indistinguishable. Over-squashing (Giovanni et al. (2024); Topping et al. (2022)) arises when long-range information is compressed, potentially leading to significant information loss. These issues are primarily caused by the convolution operation.

After reviewing related works and introducing key concepts in Section 2, we analyze the impact of graph pooling operations on over-smoothing and over-squashing in Section 3. We theoretically and empirically demonstrate that constrained pooling operators can mitigate these phenomena. Based on these findings, we propose several graph pooling approaches using Maximal Independent Sets (MIS) in Section 4. Insights into these methods, in comparison to baselines and state-of-the-art techniques, are explored in Section 5 on graph classification tasks.

This article extends Stanovic et al. (2023b) and

Stanovic et al. (2023a). In addition to the previous contributions, this work further explores the connections between pooling and over-squashing/over-smoothing from both theoretical (Section 3) and experimental (Sections 5.1 and 5.2) perspectives. Furthermore, we also provide new experiments measuring the impact of pooling on graph classification.

2 Related works

A GNN is a function that maps an input graph $\mathcal{G}^{(0)}$ to an output graph $\mathcal{G}^{(N)}$ by transforming the graph representation layer by layer. This process is typically followed by a decision network $d : \mathcal{G}^{(N)} \rightarrow y$, which performs the task-specific decision. As a result, a GNN generates a series of graph representations $(\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(N)})$. At a layer l, $\mathcal{G}^{(l)}$ is derived from $\mathcal{G}^{(l-1)}$ and is defined as $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$, where $\mathcal{V}^{(l)}$ and $\mathcal{E}^{(l)}$ correspond respectively to the set of vertices and the set of edges. Similarly, $\mathcal{G}^{(l)}$ can be expressed as $\mathcal{G}^{(l)} = (\mathbf{A}^{(l)}, \mathbf{X}^{(l)})$ where $\mathbf{A}^{(l)} \in \mathbb{R}^{n_l \times n_l}$ is the weighted adjacency matrix, $n_l = |\mathcal{V}^{(l)}|$ and $\mathbf{X}^{(l)} \in \mathbb{R}^{n_l \times f_l}$ encodes the vertices' features, with f_l the dimension of the features. Given a row u in $\mathbf{X}^{(l)}$, we denote by $x_u^{(l)}$ the attributes of the vertex $u \in \mathcal{V}^{(l)}$. On one hand, convolution operations don't modify $\mathcal{V}^{(l)}$ and $\mathcal{E}^{(l)}$ but update feature matrices $\mathbf{X}^{(l)}$. On the other hand, pooling operations aim to reduce the number of nodes and we assume that the sequence of node sets $\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(N)}$ is nested: $\mathcal{V}^{(N)} \subset \mathcal{V}^{(N-1)} \subset \cdots \subset \mathcal{V}^{(0)}$. The set of neighbors of a node v of $\mathcal{G}^{(l)}$ is denoted by $\mathcal{N}_l(v)$. The distance between two nodes in graph $\mathcal{G}^{(l)}$ is denoted by $d_{G_l}(u, v)$ and corresponds to the length of the shortest path between u and v in $\mathcal{G}^{(l)}$. This notion is extended to the distance between two sets A and B contained in $\mathcal{V}^{(l)}$, denoted as $d_{G_l}(A, B)$.

Considering the final graph $\mathcal{G}^{(N)}$ of a GNN, node prediction can be achieved by applying a MLP on each node while graph-level prediction models require first a permutation-invariant readout operator (sum, max, mean) to collapse the set of nodes into a fixed-size vector before applying a MLP.

Graph convolution The graph convolution operation is generally formalized using a MPNN (Gilmer et al. (2017)). Let's define a diffusion matrix $\mathbf{C}^{(l)}$ that encodes the set of nodes sending messages to each node in $\mathcal{G}^{(l)}$, a non-linear, element-wise activation function σ, and a learnable matrix $\mathbf{W}^{(l)}$. By restricting the messages to node representations and defining the aggregation function as a sum, a MPNN can be expressed as:

$$\mathbf{X}^{(l+1)} = \sigma(\mathbf{C}^{(l)}\mathbf{X}^{(l)}\mathbf{W}^{(l)}) \tag{1}$$

The matrix $\mathbf{C}^{(l)}$ is known as a *graph shift operator* (GSO) (Dasoulas et al., 2021). A matrix $\mathbf{C}^{(l)}$ is a GSO if and only if $\mathbf{C}_{ij}^{(l)} = 0$ for $i \neq j$ and $(i, j) \notin \mathcal{E}^{(l)}$.

Graph pooling To ensure a graph-level predictive model, a GNN needs to condense the entire graph into a fixed-size vector in a permutation-invariant manner. This operation, known as readout or global pooling, can be defined using basic statistics (sum, mean, maximum) or more intricate methods (Zhang et al., 2018). Applying global pooling thus summarizes a sophisticated graph representation into a simple vector, which leads to an inherent loss of information. To mitigate this loss, hierarchical pooling computes a gradual reduction in the size of the input graph. Moreover, using a convolution operation as a low-pass filter results in storing a reduced amount of information on the same set of nodes. This information redundancy can be counterbalanced by using hierarchical pooling (Lee et al., 2019; Ying et al., 2018).

Consider a graph $\mathcal{G}^{(l)} = (\mathbf{A}^{(l)}, \mathbf{X}^{(l)})$, and its pooled equivalent $\mathcal{G}^{(l+1)} = (\mathbf{A}^{(l+1)}, \mathbf{X}^{(l+1)})$, the relationship between them can be expressed using the reduction matrix $\mathbf{S}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$, where n_l represents the size of $\mathcal{G}^{(l)}$ and n_{l+1} represents the size of $\mathcal{G}^{(l+1)}$. $\mathbf{S}^{(l)}(i, j) \neq 0$ indicates the association between node $v_i \in \mathcal{V}^{(l)}$ to the node $v_j \in \mathcal{V}^{(l+1)}$. Given $\mathbf{S}^{(l)}$, most of pooling methods express the feature and adjacency matrices $\mathbf{X}^{(l+1)}$ and $\mathbf{A}^{(l+1)}$ of $\mathcal{G}^{(l+1)}$ as:

$$\mathbf{X}^{(l+1)} = \mathbf{S}^{(l)\top}\mathbf{X}^{(l)} \tag{2}$$

$$\mathbf{A}^{(l+1)} = \mathbf{S}^{(l)\top}\mathbf{A}^{(l)}\mathbf{S}^{(l)} \tag{3}$$

Eq. 2 defines the feature vector associated with each surviving vertex v_i as a weighted sum of the features of the vertices $v_j \in \mathcal{V}^{(l)}$, for which $\mathbf{S}_{j,i}^{(l)} \neq 0$. From Eq. 3, we get $\mathbf{A}_{i,j}^{(l+1)} = \sum_{r,s}^{n_l} \mathbf{A}_{r,s}^{(l)}\mathbf{S}_{r,i}^{(l)}\mathbf{S}_{s,j}^{(l)}$. Hence, if $\mathbf{S}^{(l)}$ is a

binary matrix, two surviving vertices i and j are adjacent in the reduced graph if there are at least two adjacent non-surviving nodes r and s, where r merges to i ($\mathbf{S}^{(l)}_{r,i} = 1$) and s merges to j ($\mathbf{S}^{(l)}_{s,j} = 1$).

Graph pooling methods can be roughly divided into two main families: cluster-based and node-drop approaches. Cluster-based pooling methods (Ying et al., 2018) define $\mathbf{S}^{(l)}$ as a cluster assignment matrix. Generally, the number of clusters is defined a priori, which prevents adapting the pooling step to the size of input graphs. When $\mathbf{S}^{(l)}$ encodes a fuzzy clustering, these methods often result in dense or even complete graphs, hence limiting the relevance of structural information.

An alternative strategy involves selecting a set of nodes to survive to the next layer. Such methods are known as Top-k or node-drop methods (Gao & Ji, 2019). The selection can be based on a learned score associated with each node, where the k nodes with the highest scores are selected for the next layer. In Gao & Ji (2019), unselected nodes are dropped, resulting in information loss and potentially leading to disconnected graphs $\mathcal{G}^{(l+1)}$. Recently, to address these issues, another strategy was proposed by Stanovic et al. (2023b) and Bacciu et al. (2023). This strategy conditions the selection of surviving nodes on satisfying a Maximal Independent Set (MIS) condition. An MIS on nodes of a graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$ is a subset $M \subset \mathcal{V}^{(l)}$, where no two nodes in M are adjacent, and every node in $\mathcal{V}^{(l)} \setminus M$ is adjacent to at least one node in M. First condition allows to guarantee a well-distributed sampling of surviving nodes while the second condition ensures that $\mathcal{G}^{(l+1)}$ is connected. A related but alternative strategy, proposed by EdgePool (Diehl et al., 2019), involves selecting edges rather than nodes.

Over-smoothing Over-smoothing refers to the phenomenon where node representations converge to vectors whose values are independent of initial node features. Over-smoothing can be measured by the mean Dirichlet energy (Rusch et al., 2023a), defined for graph $\mathcal{G}^{(l)}$ as $E(\mathbf{X}^{(l)}) = \text{Tr}((\mathbf{X}^{(l)})^T \mathbf{L} \mathbf{X}^{(l)})/|\mathcal{G}^{(l)}|$ where $\mathbf{L}$ is the Laplacian associated with $\mathbf{C}^{(l)}$ (Eq. 1). This measure evaluates the discrepancy between adjacent nodes in the graph. A low Dirichlet energy can be reached when stacking convolutional layers. As a consequence, nodes must encode the information associated with large and overlapping sets of nodes in the initial graph. When the convolutional layers act as low pass filters (Balcilar et al., 2021) such as GCN, the resulting node representations provide only a basic representation of the topology of the graph. More specifically, Li et al. (2018) have shown that, using the Kipf & Welling (2017) convolution scheme, that this type of convolution operation can be viewed as a special form of Laplacian smoothing which converges to a linear combination involving the square root of the node degree. Recently, several solutions have been proposed to mitigate over-smoothing in GNNs. For instance, GCNII (Chen et al. (2020)) employs initial residual connections and adaptive feature control to preserve the original features of nodes throughout the layers. Another approach, GraphCON-GNN (Rusch et al. (2022)) utilizes a dynamic system inspired by oscillators, where neighboring nodes influence each other without leading to complete homogeneity of their representations. Lastly, G^2-GNN (Rusch et al. (2023b)) introduces a gradient-based gating mechanism that dynamically regulates information flow.

Over-squashing Over-squashing is defined as the GNN's inability to transfer information between two distant nodes. However, as the depth of the network increases the size of receptive fields increases exponentially, causing distant node information to be squashed by information from other nodes. The distance between nodes can be measured by their commute times. Giovanni et al. (2024) proved that higher commute times lead to greater over-squashing. These commute times, or more generally communication between nodes, may be significantly influenced by bottlenecks in the graph, which are related to the negative curvature of the graph structure (Topping et al., 2022). Recent methods, based on curvature, have been proposed to address both over-smoothing and over-squashing issues (Fesser & Weber, 2024).

3 Over-smoothing and over-squashing in graph pooling

Consider a sequence of graphs $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(N)}$ produced by a GNN. For a given layer l and a graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$ we define the reduction window of $v \in \mathcal{V}^{(l)}$, denoted $RW^{(l)}(v)$, as the set of vertices in $\mathcal{V}^{(l-1)}$ used to compute the feature of v at layer l. Additionally, we assume that for any $v \in \mathcal{V}^{(l)}$, $v \in RW^{(l)}(v)$ and that reduction windows constrain the reduced graph as follows for any u, v in $\mathcal{V}^{(l)}$:

$$d_{G_{l-1}}(RW^{(l)}(u), RW^{(l)}(v)) \leq 1 \Leftrightarrow u \in \mathcal{N}_l(v)$$

In other words, two adjacent or overlapping reduction windows at layer $l-1$ induce adjacent vertices at layer l, and vice versa.

This definition creates a parent-child relationship between $v \in \mathcal{V}^{(l)}$ and its corresponding nodes in $RW^{(l)}(v)$. The transitive closure of these relationships connects any layer with the base-level graph and is referred to as a receptive field:

Definition 1 *Let $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(N)}$ denote a sequence of reduced graphs. The receptive fields at level l are defined for any vertex $v \in \mathcal{V}^{(l)}$ as:*

$$RF^{(l)}(v) = \bigcup_{u \in RW^{(l)}(v)} RF^{(l-1)}(u)$$

with $RF^{(1)}(u) = RW^{(1)}(u)$

If $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(N)}$ is built using only pooling operations (see Eq. 2 and 3), we can ensure that the set of reduction windows defined at each layer l forms a partition of $\mathcal{V}^{(l-1)}$ by ensuring that each row of $\mathbf{S}^{(l)}$ contains exactly one non-zero value. This leads to the following result (see, e.g., Jolion & Montanvert (1992)):

Remark 1 *If the set of reduction windows computed at any level l forms a partition of $\mathcal{V}^{(l-1)}$, then the set of receptive fields forms a partition of $\mathcal{V}^{(0)}$.*

Note that the above hypothesis can be weakened: If the set of reduction windows are disjoint at all levels, then the associated receptive fields are also disjoint. In such cases, the features of vertices defined at level l are computed on disjoint supports, and over-smoothing is unlikely unless the features of vertices are strongly correlated (e.g., uniform) in the base-level graph.

In classical GNNs, the sequence of graphs is built using graph convolutions with overlapping reduction windows. However, if we assume that the sequence $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(N)}$ is built by alternating a GSO with a pooling operation based on a Top-k approach (Section 2), we can still define reduction windows as the neighborhoods augmented with the central vertex. In this case, we have the following result (see Brun (2023), Proposition 7):

Proposition 1 *Given a sequence of reduced graphs $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(N)}$ built by alternating one-hop GSO convolutions with Top-k poolings, we have the following property: At any level, two vertices are non adjacent if and only if their associated receptive fields are disjoint and non-adjacent.*

This property is weaker than Remark 1. However, it shows that by alternating convolution and pooling operations, we obtain vertices computed on disjoint supports at each layer, as long as the reduced graph is not complete. Over-smoothing is also attenuated in this case.

We proposed in Stanovic et al. (2023a,b) different strategies for defining a graph hierarchy. Reduction windows produced by these strategies satisfy the following conditions at any layer l and for any vertex $w \in \mathcal{V}^{(l)}$:

$$\begin{cases} RW^{(l)}(w) = \{w\} \text{ or} \\ RW^{(l)}(w) = \{w, v_1, \ldots, v_n\} \text{ with } \forall i,\ d_{G_{l-1}}(w, v_i) = 1 \end{cases} \quad (4)$$

where $d_{G_{l-1}}(.,.)$ denotes the distance within the graph $\mathcal{G}^{(l-1)}$.

As mentioned in Section 2, over-squashing is related to the number of layers needed to combine information from two distant vertices. When using only pooling operations, this number of layers corresponds to the layer at which two given vertices can be merged into a single receptive field (Brun, 2023):

Proposition 2 *For a decimation scheme that satisfies Eq. 4 we have for any vertex w surviving at level l in the hierarchy:*

$$\forall (u, v) \in RF^{(l)}(w)^2 \quad d_{G_0}(u, v) \leq 2 * 3^l - 1$$

According to Proposition 2, two vertices u and v cannot be merged into a single receptive field before reaching level $m \geq \log_3\left(\frac{d_{G_0}(u,v)+1}{2}\right)$. Furthermore, under the assumption that pooling operations shrink all distances, it can be shown (Brun, 2023) that $m = \mathcal{O}(\log_3(d_{G_0}(u,v)))$. Thus, the number of layers required to connect two vertices is proportional to the logarithm of their distance. This contrasts with scenarios using only convolution operations, where the required number of layers increases linearly with distance. The effect of over-squashing induced by a too important number of layers is thus strongly attenuated.

When combining convolution and pooling operations, two non-adjacent vertices have disjoint receptive fields (Proposition 1). Therefore, two vertices can merge their values at layer m only if they are represented at this layer either by a single vertex (indicating they belong to the same receptive field) or by two adjacent vertices. In the first scenario, an upper bound on m is provided by Proposition 2. In the second scenario, two adjacent vertices at layer m have either adjacent or overlapping receptive fields in $\mathcal{G}^{(0)}$ (Proposition 1). Consequently, the distance between these two vertices is bounded by twice the upper bound from Proposition 2 and we obtain $m \geq \log_3\left(\frac{d_{G_0}(u,v)+2}{4}\right)$. Thus, using a combination of convolution and pooling operations leads to similar conclusions regarding over-squashing as those derived from using pooling operations alone.

4 Maximal Independent Sets and Graph Poolings

We consider a finite set $\mathcal{X}$ with a corresponding neighborhood function $\mathcal{N}$ defined on $\mathcal{X}$ such that each element of $\mathcal{X}$ is its own neighbor. A Maximal Independent Set (MIS) is a subset $\mathcal{J} \subset \mathcal{X}$ that satisfies the following two conditions:

$$\forall (x,y) \in \mathcal{J}^2 : x \notin \mathcal{N}(y) \tag{5}$$

$$\forall x \in \mathcal{X} - \mathcal{J}, \exists y \in \mathcal{J} : x \in \mathcal{N}(y) \tag{6}$$

The elements of $\mathcal{J}$ are referred to as surviving elements or survivors. Eq. 5 specifies that two adjacent elements cannot both be selected (or survive) at the same time. Eq. 6 guarantees that at least one surviving element exists in the neighborhood of each non-surviving element. From a subsampling perspective, Eq. 5 and 6 prevent both oversampling and undersampling, ensuring a uniform distribution of surviving elements throughout the structure.

A MIS can be computed using the Meer's parallel algorithm (Meer (1989)). In Meer's algorithm, a specific value v_x is assigned to each element x within the set $\mathcal{X}$. If $\mathcal{J}$ denotes a current independent set of $\mathcal{X}$, Meer's algorithm iterates over the set $\mathcal{X} - \mathcal{N}(\mathcal{J})$, where $\mathcal{N}(\mathcal{J})$ denotes the neighbors of $\mathcal{J}$, until the independent set becomes maximal. More precisely, for each iteration a vertex x is added to $\mathcal{J}$ iff v_x is maximum among x's neighbors not in $\mathcal{N}(\mathcal{J})$. The algorithm concludes when it is no longer possible to add any remaining nodes to $\mathcal{J}$ without violating Eq. 5 and 6. By construction, each element of $\mathcal{J}$ represents a local maximum at a particular iteration. Consequently, the final set produced by the algorithm can be interpreted as a maximal weight independent set.

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, if we set $\mathcal{X} = \mathcal{V}$ and $\mathcal{N}(x)$ equals to the neighborhood of vertex x in $\mathcal{G}$, Meer's algorithm produces a maximal independent vertex set. If we set $\mathcal{X} = \mathcal{E}$, and $\mathcal{N}(e)$ equals to the set of edges incident to one of the two nodes of $e \in \mathcal{E}$, Meer's algorithm produces a maximal matching of $\mathcal{G}$.

4.1 Maximal Independent Sets for Graph Pooling

In Stanovic et al. (2023b), we introduced MIVSPool, a pooling method based on a Maximal Independent Set (MIS) defined on the graph's vertex set. A similar method was proposed by Bacciu et al. (2023), using a MIS computed on the k-hops neighborhood of $\mathcal{G}^{(l)}$. These methods assign a learnable projection score to each vertex to select the most relevant ones for a specific task. However, if two adjacent vertices have identical scores, the selection becomes indeterminate. An alternative strategy is to merge similar nodes, reducing information loss from combining dissimilar vertices. Drawing from Haxhimusa (2007), we recall three edge-based learnable pooling methods from Stanovic et al.

(2023a), where $\mathbf{A}^{(l+1)}$ is obtained using Eq. 3.

Maximal Independent Edge Set (MIES) Since each level of a hierarchical pooling represents a level of abstraction from the initial graph, we define the similarity score between two vertices $x_u^{(l)}$ and $x_v^{(l)}$ at level l as: $s_{uv}^{(l)} = exp(-\|\mathbf{W}^{(l)} \cdot (x_u^{(l)} - x_u^{(l)})\|_2)$ where $\mathbf{W}^{(l)}$ is a learnable matrix. The MIES then corresponds to a maximal weight matching computed on $\mathcal{G}^{(l)}$ using this score. The resulting set of edges is denoted $\mathcal{J}^{(l)}$. This method, denoted MIESPool, can be viewed as a parallel version of EdgePool (Diehl et al. (2019)), utilizing non-oriented edges. Additionally, Landolfi (2022) proposed another parallel version of EdgePool.

Maximal Independent Edge Set with Cut (MIESCut) : The MIES method has two main drawbacks: First, many vertices that are not adjacent to $\mathcal{J}^{(l)}$ may be duplicated in the subsequent layer, resulting in a low decimation ratio (less than 50%). This does not ensure adequate abstraction at the final layer when a fixed number of layers is used. Second, the size of a maximal matching is typically much smaller than the edge set, and the attributes of the reduced graph are computed solely using the scores of the selected edges (see below). This limitation reduces the number of scores available for back-propagation, consequently diminishing the quality of the learned similarity measures.

Let $\mathcal{J}^{(l)}$ denote the maximal weighted matching at layer l. Each vertex that is not incident to $\mathcal{J}^{(l)}$ is adjacent to at least one vertex that is. To increase the decimation ratio, we associate isolated vertices with contracted ones by selecting, for each isolated vertex u, an edge e_{uv} where s_{uv} is maximal and v is incident to $\mathcal{J}^{(l)}$. This results in a spanning forest of $\mathcal{G}^{(l)}$, consisting of isolated edges, stars (trees of depth one with a central vertex), and paths of length 3. The latter represents sequences of four vertices that exhibit strong similarities to their adjacent vertices along the paths. In such a configuration, we cannot elect a surviving vertex such that the three remaining non-surviving vertices are directly adjacent to it. To satisfy the constraints stated in Eq. 4, we discard the middle edge of such paths from $\mathcal{J}^{(l)}$, thereby creating two isolated edges.

Attribute updates using MIES and MIESCut At each layer l, MIES and MIESCut define a spanning forest $\mathcal{J}^{(l)}$ of $\mathcal{G}^{(l)}$ composed of: isolated vertices (only for MIES), isolated edges, and stars (only for MIESCut). After the contraction of $\mathcal{J}^{(l)}$, the vertex attributes of $\mathcal{G}^{(l+1)}$ are updated as follows:

- For isolated vertices: The attributes defined at level l are simply duplicated at level $l+1$.
- For an isolated edge e_{uv}: Both u and v play a symmetric role, and we arbitrarily assume that u survives at the next layer. Its features are set to $x_u^{(l+1)} = s_{uv}^{(l)} x_{uv}^{(l)}$ where:

$$x_{uv}^{(l)} = (x_u^{(l)} + x_v^{(l)}) \,/\, 2 \tag{7}$$

  Note that Eq. 7 preserves the symmetry between u and v.
- For the center u of a star: We generalize Eq. 7 and set the new attribute of u as the sum of the attributes associated with each selected edge incident to u. This sum is normalized by the total of the similarity scores of all selected edges incident to u:

$$x_u^{(l+1)} = \frac{1}{\sum_{v|e_{uv}\in\mathcal{J}^{(l)}} s_{uv}^{(l)}} \sum_{v|e_{uv}\in\mathcal{J}^{(l)}} s_{uv}^{(l)} x_{uv}^{(l)} \tag{8}$$

Maximal Independent Directed Edge Set (MIDES) MIESCut performs three operations: the computation of a maximal weight matching, the assignment of nodes not incident to a selected edge, and the splitting of paths of length 3 into two isolated edges. Instead of these three steps, we propose to adopt the Maximal Independent Directed Edge Set (MIDES) reduction scheme as suggested in Haxhimusa (2007). This reduction scheme allows us avoid the splitting of paths of length 3. To do this, we need to consider oriented edges and split an edge e_{uv} into two: $e_{u\to v}$ and $e_{v\to u}$. At layer l, for each $e_{u\to v} \in \mathcal{E}^{(l)}$, their neighborhood corresponds to every edge either leaving u or v or arriving at u. More formally, the neighborhood of $\mathcal{G}^{(l)}$ is defined as:

$$\mathcal{N}^{(l)}(e_{u\to v}) = \{e_{u\to v'} \in \mathcal{E}^{(l)}\} \cup \{e_{v\to v'} \in \mathcal{E}^{(l)}\} \cup \{e_{v'\to u} \in \mathcal{E}^{(l)}\} \tag{9}$$

If we compare the neighborhoods of (9) and that of MIES, the major difference is that MIDES excludes

edges heading towards v in its neighborhood, leading to an asymmetry in selection and therefore creating stars centered on v. Indeed, computing a MIS on a directed edge set with the particular neighborhood of (9) is one of the two main principles of the MIDES algorithm.

Like our two previous methods, we assign a score to each oriented edge. We set $s_{uv}^{(l)} = \exp(-\|\mathbf{W}^{(l)} \cdot (x_u^{(l)} - x_v^{(l)}) + \mathbf{b}^{(l)}\|)$, where $\mathbf{b}^{(l)}$ is a bias term that introduces asymmetry, such that $s_{uv}^{(l)} \neq s_{vu}^{(l)}$ if $x_u^{(l)} \neq x_v^{(l)}$.

Let us consider the set of directed edges $\mathcal{D}^{(l)}$ generated through the application of a MIDES on $\mathcal{G}^{(l)}$ using our score. This set establishes a spanning forest on $\mathcal{G}^{(l)}$, comprising isolated nodes, isolated edges, and stars.

- For isolated vertices: As in MIES, the features of such vertices are copied.

- For an isolated directed edge $e_{v\to u} \in \mathcal{D}^{(l)}$: We choose u as the survivor and update its features as follows:

$$x_u^{(l+1)} = \frac{s_{vu}^{(l)} \cdot x_u^{(l)} + s_{uv}^{(l)} \cdot x_v^{(l)}}{s_{uv}^{(l)} + s_{vu}^{(l)}} \tag{10}$$

  Given $e_{v\to u} \in \mathcal{D}^{(l)}$, we notice that $s_{vu}^{(l)} > s_{uv}^{(l)}$, implying that more importance is assigned to the surviving node u. This update can be viewed as an extension of Eq. 7 using the asymmetric scores $s_{uv}^{(l)}$ and $s_{vu}^{(l)}$.

- For a star centered on u: We extend the method used for isolated edges and define the feature of u as the mean value of its incident selected edges:

$$x_u^{(l+1)} = \frac{1}{N} \sum_{v|e_{v\to u}\in\mathcal{D}^{(l)}} \frac{s_{uv}^{(l)} x_v^{(l)} + s_{vu}^{(l)} x_u^{(l)}}{s_{uv}^{(l)} + s_{vu}^{(l)}} \tag{11}$$

  with $N = |\{v \in \mathcal{V}^{(l)} \mid e_{v\to u} \in \mathcal{D}^{(l)}|$.

Let us note that the attribute updates of MIVS, MIES, MIESCut, and MIDES can be performed in parallel using Eq.2 through an appropriate setting of the matrix $\mathbf{S}^{(l)}$ (Stanovic et al. (2023b,a)).

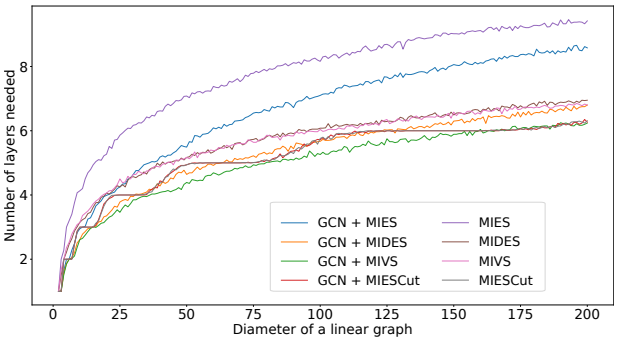

Figure 1: Number of layers required to combine features of distant vertices.

5 Experiments

5.1 Over-squashing

To measure the over-squashing phenomenon in GNNs based on convolution and/or pooling operations, we generate linear graphs with varying diameters. We assign the features $(1, 0)$ and $(0, 1)$ to the two degree-1 vertices, while all other vertices are set to $(0, 0)$. Our goal is to observe the number of layers required for a vertex to have both non-null features (with a threshold set at 10^{-5})

We consider the following methods: GCN convolution (Kipf & Welling (2017)), MIVSPool (Stanovic et al. (2023b)), MIESPool, MIESCutPool, and MIDESPool (Section 4), as well as alternating GCN convolution and pooling (GCN+MIESPool, GCN+MIVSPool, GCN+MIESCutPool, and GCN+MIDESPool). Note that Top-k methods, which discard non-selected vertices (Section 2), may eliminate one of the degree-1 vertices, thereby preventing feature combination. Dense pooling methods (e.g., Ying et al. (2018)) create a complete graph, thereby providing such a vertex at the first pooling layer.

The number of layers required for GCN to combine the features of degree-1 vertices is not shown in Fig. 1 to avoid overwhelming other curves. However, we observe a linear relationship between the required number of layers and the graph diameter. For all other methods, the logarithmic relationship between node distance and the required layers, as stated in Section 3, is confirmed.

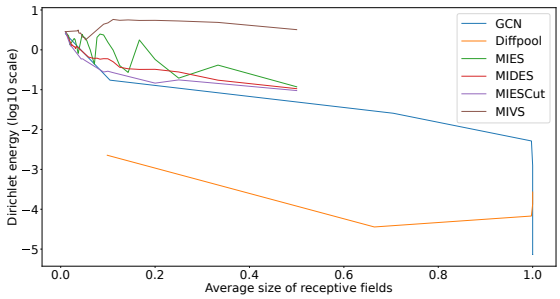

Figure 2: Dirichlet energy computed using GCN and pure pooling methods on random graphs.

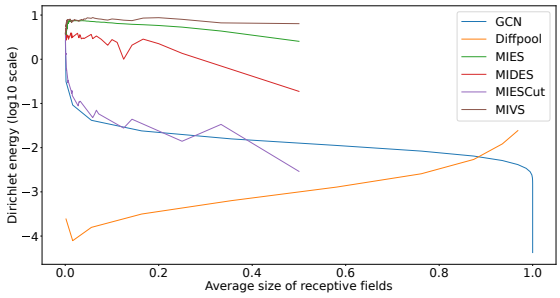

Figure 3: Dirichlet energy computed using GCN and pure pooling methods on Cora dataset.

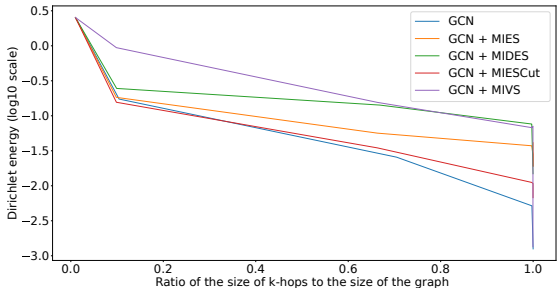

Figure 4: Dirichlet energy computed using alternating GCN and pooling methods on random graphs.

5.2 Over-smoothing

To evaluate the over-smoothing effect, we generate 100 random binomial graphs, each with 100 vertices and a fixed mean degree of 10. Vertices are assigned random vectors of dimension 32, drawn from a continuous uniform distribution in $[0, 1]$. For each graph, we apply a GNN with up to 50 layers and measure the mean Dirichlet Energy (Section 2) at each layer.

Different pooling methods exhibit varying decimation ratios, and GNNs abstract graph information at different

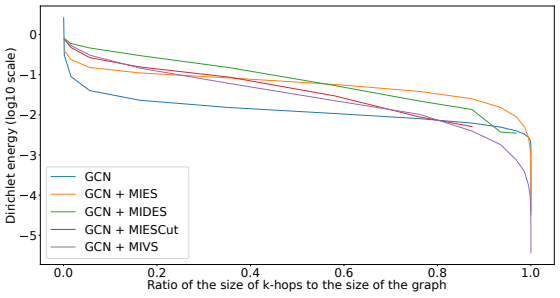

Figure 5: Dirichlet energy computed using alternating GCN and pooling on Cora dataset.

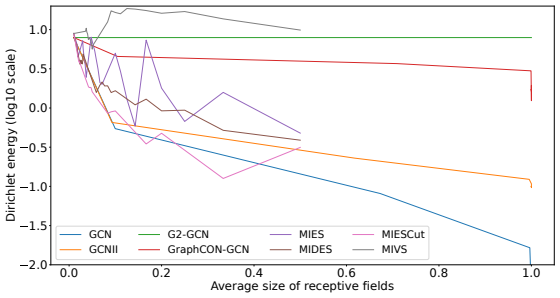

Figure 6: Comparison on random graphs between pooling methods and graph convolution methods designed to reduce over-smoothing.

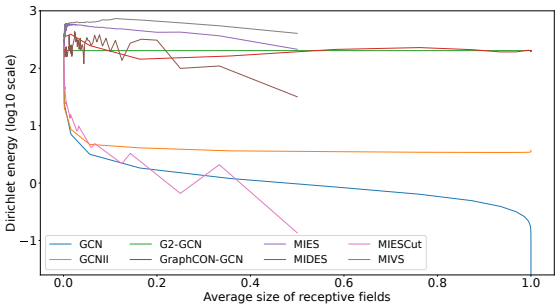

Figure 7: Comparison on Cora dataset between pooling methods and graph convolution methods designed to reduce over-smoothing.

rates. To compare the Dirichlet Energy across different GNNs, we plot it according to the ratio of the mean size of the receptive fields at different layers to the graph size. For pure convolution methods, we calculate the ratio of the mean size of k-hops at layer k to the graph size. We apply the same approach for DiffPool (Ying et al. (2018)), which generates a dense aggregation matrix and receptive fields that approximate the complete graph at the first iteration. All computed values are normalized between 0 and 1 by dividing by the graph size. Additionally, to compare pooling strategies rather than aggregation functions, we standardize the aggregation function of all pooling methods to a simple mean.

Fig. 2 displays the mean value (averaged over 100 graphs) of the mean Dirichlet Energy (log scale) for GCN (Kipf & Welling (2017)) and pure pooling methods. We observe an abrupt drop in energy for the GCN method when the mean size of the hops reaches 100% of the vertex set. The MIVS curve shows a slight decrease as the mean size of the receptive fields increases. Edge-based pooling methods (MIDES, MIESCut, and MIES) exhibit a more significant decrease, with MIESCut having the lowest curve, yet remaining above GCN. These decreases may be partially attributable to the computation of larger means based on data from the same distribution. DiffPool generates an almost connected graph with highly overlapping receptive fields, which accounts for its abrupt drop in Dirichlet Energy (Section 3).

Fig. 3 illustrates the same curves computed on the largest connected component of the Cora graph (McCallum et al., 2000), which contains 2485 nodes and 10138 edges. Following the approach proposed in Rusch et al. (2023a), we replace the original one-hot features with random vectors of dimension 32. The curves for MIVS, MIES, and MIDES exhibit a slight decrease relative to the size of the receptive fields but remain above the GCN curve. The MIESCut curve is positioned below the others and is close to GCN, even dipping below it for the ratio $x = 0.5$. This occurs because MIESCut does not permit isolated vertices, thereby enforcing convergence to the mean and reducing Dirichlet Energy. As previously noted, DiffPool's Dirichlet Energy remains below all other curves.

Fig. 4 and 5 present the same plots for random graphs and the Cora dataset, utilizing alternating convolution (Kipf & Welling (2017)) and pooling operations at each layer. Since all methods incorporate pooling operations, the curves are plotted based on the mean size of the k-hops. In the case of small random graphs (Fig. 4), all methods except MIESCut mitigate over-smoothing compared to GCN. The slightly lower Dirichlet Energy for MIESCut at $x = 0.1$ is likely due to insufficient decimation to counterbalance the effects of double averaging at each layer.

On the large Cora graph (Fig. 5), all pooling methods except MIVS mitigate the over-smoothing induced by GCN. For large graphs, MIVS generates stars within reduced graphs (Haxhimusa (2007)), drastically reducing the decimation ratio, which becomes insufficient to counterbalance the aggregation effects of convolution operations. This phenomenon is unique to MIVS and does not occur in small to medium graphs, such as the random graphs (Fig. 4).

Fig. 6 and 7 compare our pooling methods and GCN with convolution techniques that mitigate over-smoothing, using random graphs and the Cora dataset, respectively. The methods compared include GCNII (Chen et al., 2020), GraphCON-GCN (Rusch et al., 2022) and G^2-GCN (Rusch et al., 2023b). Note that GCNII requires the feature dimension of a node to match the number of vertices in the graph, and all methods were re-evaluated under this condition. In both figures, G^2-GCN and GraphCON-GCN remain unaffected by over-smoothing. In contrast, GCNII is more sensitive to over-smoothing but stabilizes its Dirichlet energy as the size of k-hops increases. Additionally, the Dirichlet energy of MIVS is higher than that of convolution methods designed to mitigate over-smoothing. Except for MIESCut, the energies of the other pooling methods exceed those of GCNII or are comparable to G^2-GCN and GraphCON-GCN. This highlights the effectiveness of our pooling methods in mitigating the effects of over-smoothing.

5.3 Graph classification

We evaluate our contributions on five graph classification datasets. D&D (Dobson & Doig, 2003) involves classifying proteins as either enzymes or non-enzymes. NCI109 (Wale et al., 2008) contains small molecules associated with their activity against ovarian cancer. The datasets REDDIT-BINARY, REDDIT-5K, and REDDIT-

Table 1: Average classification accuracies of ours methods. Highest accuracies are highlighted in **bold** while the second highest accuracies are in blue. $\pm$ denotes the 95% confidence interval of classification accuracy.

Ours Methods	**D&D**	**NCI109**	**REDDIT-BINARY**	**REDDIT-5K**	**REDDIT-12K**
GCN+MIVSPool	76.35 ± 2.09	72.09 ± 1.27	$\mathbf{88.73 \pm 4.43}$	52.17 ± 1.90	46.50 ± 3.04
GCN+MIESPool	77.17 ± 2.33	$\mathbf{73.43 \pm 2.27}$	88.08 ± 4.55	$\mathbf{54.40 \pm 1.27}$	47.33 ± 3.37
GCN+MIESCutPool	$\mathbf{77.74 \pm 2.85}$	71.81 ± 2.19	86.47 ± 4.57	53.45 ± 0.91	$\mathbf{47.51 \pm 3.05}$
GCN+MIDESPool	76.52 ± 2.21	71.49 ± 1.77	88.40 ± 4.74	53.62 ± 1.35	46.51 ± 3.40
Linear+MIVSPool	74.11 ± 2.69	70.13 ± 2.04	86.85 ± 5.17	53.12 ± 1.39	45.96 ± 2.95
Linear+MIESPool	75.95 ± 3.14	68.86 ± 2.78	87.38 ± 4.33	52.74 ± 1.56	45.79 ± 2.95
Linear+MIESCutPool	75.72 ± 2.77	68.95 ± 1.92	83.62 ± 6.83	53.42 ± 1.11	46.39 ± 3.37
Linear+MIDESPool	75.75 ± 1.89	66.59 ± 2.06	85.77 ± 5.21	52.64 ± 1.33	44.89 ± 3.08

Table 2: Average classification accuracies through various pooling methods in comparison with our best method. Highest accuracies are highlighted in **bold** while the second highest accuracies are in blue. $\pm$ denotes the 95% confidence interval of classification accuracy.

Methods	**D&D**	**NCI109**	**REDDIT-BINARY**	**REDDIT-5K**	**REDDIT-12K**
Baseline	76.29 ± 2.33	73.37 ± 1.90	87.07 ± 4.72	52.34 ± 2.97	47.45 ± 2.89
GCN+gPool	75.61 ± 2.74	67.78 ± 1.61	84.37 ± 7.82	50.41 ± 1.44	44.00 ± 2.92
GCN+SagPool	76.15 ± 2.88	68.40 ± 1.59	85.63 ± 6.26	50.84 ± 0.93	44.64 ± 2.82
GCN+EdgePool	72.59 ± 3.59	72.69 ± 2.47	87.25 ± 4.78	48.90 ± 2.12	43.25 ± 2.45
Our best method	$\mathbf{77.74 \pm 2.85}$	$\mathbf{73.43 \pm 2.27}$	$\mathbf{88.73 \pm 4.43}$	$\mathbf{54.40 \pm 1.27}$	$\mathbf{47.51 \pm 3.05}$

12K (Yanardag & Vishwanathan, 2015) comprise social graphs of online Reddit discussions, categorized into two, five, and eleven community classes, respectively. The model architecture and training procedure follow those described in Stanovic et al. (2023a).

We evaluate the performance of our pooling methods on classification tasks in Table 1 by testing two configurations: one alternating between convolution and pooling, and one with a simple linear layer followed by pooling. The highest classification accuracy is consistently achieved with pooling methods combined with convolution, demonstrating the added value of convolutions. However, the difference between the two configurations often falls below the standard deviation. This suggests, firstly, that there is some redundancy between the two successive aggregation steps performed by convolution and pooling. Secondly, we anticipate that further work on the aggregation step of pooling methods may improve the classification accuracy of pure pooling methods, aiming to achieve results comparable to those of the combined convolution and pooling approaches.

In Table 2, we compare the best accuracy achieved by one of our methods (from Table 1) with four state-of-the-art methods for each dataset: Baseline (K blocks of GCN), gPool (Gao & Ji, 2019), SagPool (Lee et al., 2019), and EdgePool (Diehl et al., 2019). We observe that the baseline results are quite good without pooling, underscoring the challenge of designing effective pooling methods. Alternative pooling methods often yield lower accuracy than GCN, with Top-k methods leading to significant information loss. For each dataset, the highest accuracy is attained by one of our GCN+pooling methods, surpassing the baseline. On the D&D dataset, all our GCN+pooling methods outperform GCN, whereas only one method does so on the NCI109 dataset, likely due to the small size of its graphs.

6 Conclusion

This paper presents innovative pooling methods based on maximal independent sets for GNNs, effectively addressing the limitations of traditional pooling approaches. These methods mitigate several drawbacks of GNNs by elucidating the relationships between receptive fields at various levels of hierarchical pooling. Our theoretical and empirical studies validate the effectiveness of these methods, marking a significant advancement in the field of GNNs and opening new avenues for future research.

Acknowledgements: This work was supported by French ANR grant #ANR-21-CE23-0025 CoDeGNN and was conducted using HPC resources from GENCI-IDRIS (Grant 2022-AD011013595) as well as computing resources from CRIANN (Grant 2022001, Normandy, France).

References

Bacciu, D., Conte, A., & Landolfi, F. (2023). Generalizing downsampling from regular data to graphs. *AAAI*, *37*, 6718–6727.

Balcilar, M., Guillaume, R., Héroux, P., Gaüzère, B., Adam, S., & Honeine, P. (2021). Analyzing the expressive power of graph neural networks in a spectral perspective. In *ICLR*.

Brun, L. (2023). *Pooling properties within the Graph Neural network framework*. Technical Report ENSICAEN. URL: `https://www.normastic.fr/wp-content/uploads/2024/01/smoothing_squashing.pdf`.

Chen, M., Wei, Z., Huang, Z., Ding, B., & Li, Y. (2020). Simple and deep graph convolutional networks. In *ICML* (pp. 1725–1735). PMLR volume 119.

Dasoulas, G., Lutzeyer, J. F., & Vazirgiannis, M. (2021). Learning parametrised graph shift operators. In *ICLR*. OpenReview.net.

Diehl, F., Brunner, T., Le, M. T., & Knoll, A. (2019). Towards graph pooling by edge contraction. In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*.

Dobson, P. D., & Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *J. of molecular biology*, *330*, 771–783.

Fesser, L., & Weber, M. (2024). Mitigating over-smoothing and over-squashing using augmentations of forman-ricci curvature. In *Proc. of the Second Learning on Graphs Conference* (pp. 19:1–19:28). PMLR volume 231.

Gao, H., & Ji, S. (2019). Graph u-nets. In *ICML* (pp. 2083–2092). PMLR volume 97.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *ICML* (pp. 1263–1272). PMLR volume 70.

Giovanni, F. D., Rusch, T. K., Bronstein, M., Deac, A., Lackenby, M., Mishra, S., & Veličković, P. (2024). How does over-squashing affect the power of GNNs? *Transactions on Machine Learning Research*, .

Haxhimusa, Y. (2007). *The structurally Optimal Dual Graph Pyramid and its application in image partitioning* volume 308. IOS Press.

Jolion, J., & Montanvert, A. (1992). The adaptive pyramid: A framework for 2d image analysis. *CVGIP Image Underst.*, *55*, 339–348.

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *ICLR*.

Landolfi, F. (2022). Revisiting edge pooling in graph neural networks. In *ESANN*.

Lee, J., Lee, I., & Kang, J. (2019). Self-attention graph pooling. In *ICML* (pp. 3734–3743). PMLR volume 97.

Li, Q., Han, Z., & Wu, X. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. (pp. 3538–3545). volume 32.

McCallum, A. K., Nigam, K., Rennie, J., & Seymore, K. (2000). Automating the construction of internet portals with machine learning. *Inf. Retr.*, *3*, 127–163.

Meer, P. (1989). Stochastic image pyramids. *Computer Vision, Graphics, and Image Processing*, *45*, 269–294.

Rusch, T. K., Bronstein, M. M., & Mishra, S. (2023a). A survey on oversmoothing in graph neural networks. *CoRR*, *abs/2303.10993*.

Rusch, T. K., Chamberlain, B., Rowbottom, J., Mishra, S., & Bronstein, M. M. (2022). Graph-coupled oscillator networks. In *ICML* (pp. 18888–18909). PMLR volume 162.

Rusch, T. K., Chamberlain, B. P., Mahoney, M. W., Bronstein, M. M., & Mishra, S. (2023b). Gradient gating for deep multi-rate learning on graphs. In *ICLR*. OpenReview.net.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Trans. Neural Networks*, *20*, 61–80.

Stanovic, S., Gaüzère, B., & Brun, L. (2023a). Maximal independent sets for pooling in graph neural networks. In *GbRPR* (pp. 113–124). Springer volume 14121 of *LNCS*.

Stanovic, S., Gaüzère, B., & Brun, L. (2023b). Maximal independent vertex set applied to graph pooling. In *S+SSPR* (pp. 11–21). Springer volume 13813 of *LNCS*.

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., & Bronstein, M. M. (2022). Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*.

Wale, N., Watson, I. A., & Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, *14*, 347–375.

Yanardag, P., & Vishwanathan, S. (2015). Deep graph kernels. In *Int. Conf. on Knowledge Discovery and Data Mining* (pp. 1365–1374). ACM.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *NeurIPS* (pp. 4805–4815).

Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. *AAAI*, *32*, 4438–4445.