Maximal Independent Vertex Set applied to Graph Pooling

Stevan Stanovic, Benoit Gaüzère, Luc Brun

Maximal Independent Vertex Set applied to Graph Pooling

Stevan Stanovic[1] [0000−0001−9656−2080], Benoit Gaüzère[2] [0000−0001−9980−2641], and Luc Brun[1] [0000−0002−1658−0527]

[1] Normandie Univ, ENSICAEN, CNRS, UNICAEN, GREYC UMR 6072, 14000 Caen, France
{stevan.stanovic, luc.brun}@ensicaen.fr
[2] Normandie Univ, INSA de Rouen, Univ. rouen, Univ. Le Havre, LITIS EA 4108, 76800 Saint-Étienne-du-Rouvray, France
benoit.gauzere@insa-rouen.fr

Abstract. Convolutional neural networks (CNN) have enabled major advances in image classification through convolution and pooling. In particular, image pooling transforms a connected discrete grid into a reduced grid with the same connectivity and allows reduction functions to take into account all the pixels of an image. However, a pooling satisfying such properties does not exist for graphs. Indeed, some methods are based on a vertex selection step which induces an important loss of information. Other methods learn a fuzzy clustering of vertex sets which induces almost complete reduced graphs. We propose to overcome both problems using a new pooling method, named MIVSPool. This method is based on a selection of vertices called surviving vertices using a Maximal Independent Vertex Set (MIVS) and an assignment of the remaining vertices to the survivors. Consequently, our method does not discard any vertex information nor artificially increase the density of the graph. Experimental results show an increase in accuracy for graph classification on various standard datasets.

Keywords: Graph Neural Networks · Graph Pooling · Graph Classification · Maximal Independant Vertex Set.

1 Introduction

Convolutional neural networks (CNN) have enabled major advances in image classification. An image can be defined as a connected discrete grid and this property enables to define efficient convolution and pooling operations. Nevertheless, social networks, molecules or traffic infrastructures are not represented by a grid but by graphs. Convolution and pooling have been adapted to these structured data into Graph Neural Networks (GNN) [11]. The adaptation of convolution to graphs can be performed by using learned aggregation functions which combine the value of each vertex with the ones of its neighborhood [6,7]. For graph pooling, the adaptation is mainly performed by selecting vertices [2,5,8,15] or by learning a fuzzy clustering [13].

The work reported in this paper was supported by French ANR grant #ANR-21-CE23-0025 CoDeGNN

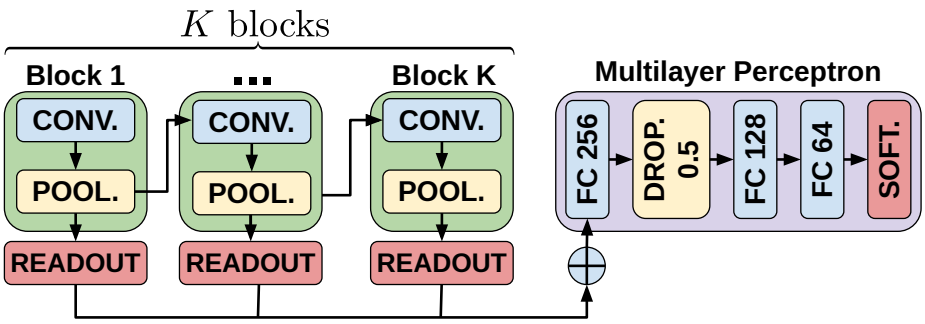


Fig. 1: General architecture of our GNN. Each block is composed of a convolution layer followed by a pooling layer. Features learned after each block are combined to have several levels of description of the graph.

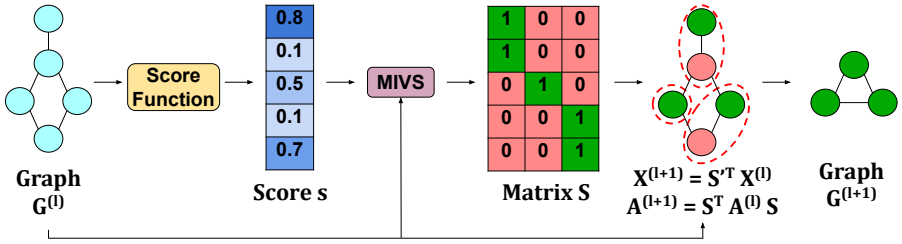


Fig. 2: Proposed graph pooling. Each node is associated to a score (Section 3.2). Based on this score, a MIVS is computed from which a reduction matrix S is derived. Applying S to both feature and structure leads to a reduced graph $G^{(l+1)}$.

In this paper, we detail graph convolution and pooling as well as the construction of the reduced graph in Section 2. Section 3 presents our method (Fig. 2) and explains its differences from other pooling methods. We present in Section 4 our experiments where we study different heuristics to select surviving vertices and the mean complexity of our pooling algorithm. We finally compare our method to other pooling strategies on standard datasets using an unified architecture (Fig. 1) in Section 4.

2 Related Work

GNNs are neural networks applied to graphs. Inspired by CNNs, they are like the latter based on convolution and pooling operations. Let us note that convolution operation should be permutation equivariant. Intuitively, this condition states that a convolution operation may be permuted with a permutation of the graph's nodes.

Convolution operations diffuse the information by a message passing mechanism which allows to learn a representation for each node by aggregating the information of their n-hop neighborhood. Optimized according to a particular task, the resulting nodes' features can be used as input of a node classifier or regressor. According to [1], current aggregation functions correspond mainly to low-pass filters.

2.1 Graph Pooling

Considering graph level tasks like graph classification, we need to compute a graph representation as a fixed sized vector by summing node's representation. This operation is called global pooling. Global pooling realizes an aggregation of all the vertices of a graph and summarizes it in a single vector. Such aggregation must be permutation invariant and is usually performed using basic operators like a sum, mean, maximum or more complex ones [14].

However, global pooling performed on all the vertices of a graph leads to sum up an unbounded amount of information (proportional to the graph's size) into a fixed size vector and thus potentially induces a large amount of information loss. Many authors [8,13] have proposed to decompose GNNs in several steps alternating convolution and pooling to reduce the graph size while averaging vertices values. Such methods are called hierarchical pooling. The choice of the number of vertices for the reduced graph can be fixed or adapted according to the original graph's size with a ratio. The hierarchical pooling, like the convolution operation should be permutation equivariant.

Notations. For any pooling layer l, we consider a graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$ where $\mathcal{V}^{(l)}$ and $\mathcal{E}^{(l)}$ are respectively the set of vertices and the set of edges of the graph. Let $n_l = |\mathcal{V}^{(l)}|$, we can define $\mathbf{A}^{(l)} \in \mathbb{R}^{n_l \times n_l}$ the adjacency matrix associated to the graph $\mathcal{G}^{(l)}$ where $\mathbf{A}^{(l)}_{ij} = 1$ if it exists an edge between the vertices i and j, 0 otherwise. We also note $\mathbf{X}^{(l)} \in \mathbb{R}^{n_l \times f_l}$ the feature matrix of the graph $\mathcal{G}^{(l)}$ where f_l is the dimension of nodes' attributes.

Construction of the set of surviving vertices. It exists multiple methods to reduce the size of a graph within the GNN framework. However, most of these methods lead to the construction of a reduction matrix $S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ where n_l and n_{l+1} are respectively the sizes of the original and the reduced graph. This matrix is used to define the attributes and the adjacency matrix of the reduced graph. Each surviving vertex i contributing to its own cluster, we suppose that $S^{(l)}_{i,i} = 1$.

Construction of attributes and adjacency matrix. The reduction matrix $S^{(l)}$ is the basis of the construction of the reduced graph. Based on $S^{(l)}$, the following two equations allow this construction:

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

This last equation defines the attribute of each surviving vertex v_i as a weighted sum of the attributes of the vertices v_j of $G^{(l)}$ such that $S^{(l)}_{ji} \neq 0$.

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

Equation (2) can be rewritten as follows for any pair of surviving vertex (i, j):

$$(S^{(l)\top} A^{(l)} S^{(l)})_{i,j} = \sum_{k,l}^{n_l} A^{(l)}_{k,l} S^{(l)}_{k,i} S^{(l)}_{l,j} \tag{3}$$

Two surviving vertices are therefore adjacent in the reduced graph if they are adjacent in the initial graph ($A^{(l)}_{i,j} = S^{(l)}_{i,i} = S^{(l)}_{j,j} = 1$). Moreover, surviving vertices i and j are

adjacent in the reduced graph if it exists a pair of non-surviving adjacent vertices (k, l) assigned respectively to i and j ($A^{(l)}_{k,l} = S^{(l)}_{k,i} = S^{(l)}_{l,j} = 1$).

Families of methods. Pooling methods can be divided in two families. First family consists of methods based on the selection of surviving vertices on a given criteria. This criteria can be the result of a combinatorial algorithm [2] or a learning step like in Top-*k* methods [5,8]. The second family regroups methods based on a node's clustering as in DiffPool [13]. Each cluster is associated to a surviving vertex.

Methods of the second group use a fixed number of clusters. Hence, learning $S^{(l)}$ does not allow these methods to take into account the variable size and topology of the graphs. Moreover, due to training, such methods produce dense matrices $S^{(l)}$ with few nonzero values. Equation (3) shows that the matrix $A^{(l+1)}$ is then dense and the corresponding graph has a density close to 1 (i.e. will be a complete graph or almost complete graph). Consequently, the structure of the graph is not respected. We say that $S^{(l)}$ is *complete*.

For Top-*k* methods, we define the reduction matrix $S^{(l)}$ as the restriction of the identity matrix I_{n_l} to the column indices idx corresponding to the surviving vertices:

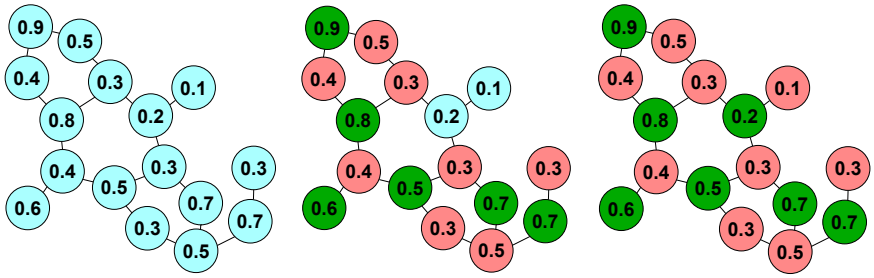
$$S^{(l)} = [I_{n_l}]_{:,idx} \tag{4}$$

Given a surviving vertex i in the original graph, we have thus $S^{(l)}_{i,\phi(i)} = 1$, where $\phi(i)$ denotes the column index of i in the reduced matrix $S^{(l)}$. All other entries of $S^{(l)}$ are set to 0. The matrix $S^{(l)}$ is called *selective* since it selects the attributes of the surviving vertices and removes those of non-surviving vertices. Moreover, in this case, rows of $S^{(l)}$ corresponding to non-surviving vertices are equal to 0 and two surviving vertices will be adjacent if and only if they were adjacent before the reduction. This last point may induce the creation of disconnected reduced graphs. Moreover, the drop of non-survivors' features leads to an important loss of information. Let us note that MVPool [15] increases the density of the graph by considering power 2 or 3 of the adjacency matrix in order to limit the disconnections of the reduced graph. It additionally adds edges to the reduced graph through an additional layer called Structure Learning.

An alternative solution consists to drop Equation (2) and to perform a Kron reduction [2] in order to connect all pairs of surviving vertices adjacent to a same removed vertex. This reduction increases the density of the graph and a sparsification step is required. This last point can creates a disconnected reduced graph. Moreover, the time complexity of the Kron reduction is approximately $\mathcal{O}((\frac{n_l}{2})^3)$, due to the inversion of a part of the Laplacian matrix of $\mathcal{G}^{(l)}$.

Let us finally note that it exists a graph pooling method which uses an approximation of a Maximum Independent Vertex Set called MEWIS [10]. The layer used to compute this approximation significantly increases the complexity of the whole GNN. Moreover, the approximated result provided by this layer does not guarantee that the resulting set of selected vertices is even maximal.

Unlike other methods, we propose to preserve the structure of the original graph as well as the attribute information. We satisfy these properties thanks to a selection of surviving vertices distributed equally on the graph and with an assignment of non-surviving vertices to surviving vertices. We named our pooling method MIVSPool.



(a) original graph (b) after first iteration (c) after second iteration

Fig. 3: Evolution of MIVS algorithm on a graph from the NCI1 dataset (Section 4.1). Number inside each vertex corresponds to its score s. Candidate, Survivor and Non-Survivor vertices are respectively denoted by: ○, ●, ●.

3 Proposed Method

3.1 Maximal Independent Vertex Set (MIVS)

Before describing the algorithm, we introduce the notion of Maximal Independent Set and show how this notion can be applied to graph vertices.

Maximal Independent Set. Let $\mathcal{X}$ be a finite set and $\mathcal{N}$ a neighborhood function. A subset $\mathcal{J}$ of $\mathcal{X}$ is independent if:

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

$\mathcal{J}$ is a subset of $\mathcal{X}$ such that for any $(x, y) \in \mathcal{J}^2$, x and y are not neighbors. The elements of $\mathcal{J}$ are called the surviving elements.

An independent set $\mathcal{J}$ is said to be maximal when no element can be added to it without breaking the independence property, i.e., when we have:

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

Equation (6) states that each non-surviving element has to be in the neighborhood of at least one element of $\mathcal{J}$. The elements of $\mathcal{J}$ are denoted as survivors.

Using Equation (5) and (6), we have a Maximal Independent Set. We note that it is a maximal but not necessarily a maximum. Indeed, our Maximal Independent Set $\mathcal{J}$ is not necessarily those whose cardinality is maximum with respect to all Maximal Independent Sets of $\mathcal{X}$.

If we interpret the construction of a Maximal Independent Set as a subsampling operation, Equation (5) can be interpreted as a condition preventing the oversampling (two adjacent vertices cannot be simultaneously selected) which thus guarantees an uniform distribution of survivors. Conversely, Equation (6) prevents subsampling: Any non-selected vertex is at a distance 1 from a surviving vertex.

Maximal Independent Vertex Set. A Maximal Independent Vertex Set (MIVS) [9] of a graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$ is therefore a Maximal Independent Set where the neighborhood

is deduced from the edge set $\mathcal{E}^{(l)}$. Adapting Equation (5) and (6), we select surviving vertices $\mathcal{V}^{(l+1)}$ as described below:

$$\forall (v, v') \in (\mathcal{V}^{(l+1)})^2 : (v, v') \notin \mathcal{E}^{(l)} \tag{7}$$

$$\forall v \in \mathcal{V}^{(l)} - \mathcal{V}^{(l+1)}, \exists v' \in \mathcal{V}^{(l+1)} : (v, v') \in \mathcal{E}^{(l)} \tag{8}$$

Equations (7) and (8) define the MIVS procedure and state that two surviving vertices cannot be neighbors and a non-surviving vertex must have at least one survivor in its neighborhood. Nevertheless, these two equations don't explain how to select these vertices.

Meer's Algorithm. A simple method has been proposed by Meer [9] using an iterative procedure assigning a uniform random variable $\mathcal{U}([0, 1])$ to each vertex. In this procedure, each surviving vertex corresponds at a local maximum of the random variable with respect to its neighbors. According to Equation (7), vertices adjacent to these survivors are labeled as non-survivors. Other vertices (not yet labeled) are labeled candidates and the algorithm iterates the selection of surviving and non-surviving vertices on this reduced vertex set (Fig. 3). The algorithm convergence is guaranteed since, at each iteration, at least one candidate is labeled as a survivor. For each vertex v_i, its random variable is denoted by x_i while Booleans p_i and q_i are $True$ if v_i is respectively a survivor and a candidate. The stopping criterion of the algorithm is obtained when all q_i are $False$, i.e. when Equation (8) is satisfied. At the initialisation of the algorithm, all p_i are $False$ and all q_i are $True$ and, for each iteration k, variables $p_i^{(k)}$ and $q_i^{(k)}$ are updated by:

$$\begin{aligned} p_i^{(k+1)} &= p_i^{(k)} \vee (q_i^{(k)} \wedge (x_i = \max\{x_j | (v_i, v_j) \in \mathcal{E}^{(l)} \wedge q_j^{(k)}\})) \\ q_i^{(k+1)} &= \wedge_{j|(v_i,v_j)\in\mathcal{E}^{(l)}} \overline{p}_j^{(k+1)} \end{aligned} \tag{9}$$

In other words, a survivor at iteration $k + 1$ is a survivor at iteration k or is a candidate whose random variable (x_i) is greater than the ones of its candidate neighbors. A vertex is candidate at iteration $k + 1$ if it is not adjacent to a survivor. We note that the neighborhood includes the central vertex. This procedure only involves local computations and is therefore parallelizable.

3.2 Adaptation of MIVS to deep learning

Scoring system. Using Meer's algorithm [9], the uniform random variable $\mathcal{U}([0, 1])$ plays an important role in the selection of the surviving vertex set. We propose to modify this variable so that x_i used in the algorithm is learnt and represents the relevance of vertex v_i. We obtain this last property by using an attention mechanism like in SagPool [8] where a score vector $s \in \mathbb{R}^{n_l \times 1}$ is returned by a GCN [7]. Using this score in our MIVS, we select a set of surviving vertices $\mathcal{V}^{(l+1)}$ corresponding to local maxima of the function of interest encoded by s. An uniform distribution of vertices on the graph is guaranteed by the computation of the MIVS.

Assignment of Non-Surviving Vertices. In order to construct our reduction matrix $S^{(l)}$ and take all vertices information into account, we need to assign non-surviving

vertices. As a reminder, Equation (8) states that each non-survivor has at least a surviving neighbor. Assuming that the score function encodes the relevance of each vertex, we assign each non-surviving vertex to its surviving neighbor with the highest score. We obtain a reduction matrix $S^{(l)}$ with n_{l+1} clusters corresponding to surviving vertices:

$$\begin{aligned} &\forall v_j \in \mathcal{V}^{(l)} - \mathcal{V}^{(l+1)}, \exists! v_i \in \mathcal{V}^{(l+1)} | \mathcal{S}_{ji}^{(l)} = 1 \\ &\text{with } i = argmax(s_k, (v_k, v_j) \in \mathcal{E}^{(l)} \wedge p_k^N) \end{aligned} \tag{10}$$

where N is the number of iterations required to have a MIVS and s the score vector.

Construction of reduced attributes and adjacency matrix. The construction of attributes of the reduced graph $\mathcal{G}^{(l+1)}$ is obtained thanks to an average weighted by s from the set of aggregated vertices to the surviving vertices:

$$X_i^{(l+1)} = \frac{1}{\sum_{j|S_{ji}^{(l)}=1} s_j} \sum_{j|S_{ji}^{(l)}=1} s_j X_j^{(l)} \tag{11}$$

Equation (11) allows to take into account the importance of each vertex in the computation of the attributes of the reduced graph and put more attention on vertices with a high score. As a consequence the learnt vector s can be interpreted as a relevance value. This operation can be achieved by a transformation similar to Equation (1) by substituting $S^{(l)}$ by the matrix $S^{(l)'} = D_2\, D_1\, S^{(l)}$ with $D_1 = diag(s)$ and $D_2 = diag(\frac{1}{\mathbf{1}^\top D_1 S^{(l)}})$. The construction of the reduced adjacency matrix $A^{(l+1)}$ is obtained thanks to Equation (2).

Relaxation of MIVSPool. On some highly dense graphs, MIVSPool may provide a decimation ratio lower than 0.5. In order to correct this point, we additionally introduce MIVSPool$_{comp.}$ which is a MIVSPool where we force the addition of surviving vertices using a Top-*k* so that the pooling ratio remains always equal to 0.5. Note that Equation (7) is then no more valid while Equation (8) still holds.

4 Experiments

4.1 Datasets

To evaluate our MIVSPool, we test it on a benchmark of four standard datasets: D&D [4], PROTEINS [3,4], NCI1 [12] and ENZYMES [3]. The statistics of datasets are reported on Table 1. D&D and PROTEINS describe proteins and the aim is to classify them as enzyme or non-enzyme. Nodes represent the amino acids and two nodes are connected

Dataset	**#Graphs**	**#Classes**	**Avg** $\lvert\mathcal{V}\rvert$	**Avg** $\lvert\mathcal{E}\rvert$
D&D	1178	2	284 ± 272	715 ± 694
PROTEINS	1113	2	39 ± 46	72 ± 84
NCI1	4110	2	29 ± 13	32 ± 14
ENZYMES	600	6	33 ± 15	62 ± 26

Table 1: Statistics of datasets

by an edge if they are less than 6 Ångström apart. NCI1 describes molecules and the purpose is to classify them as cancerous or non-cancerous. Each vertex stands for an atom and edges between vertices represent bonds between atoms. ENZYMES is a dataset of protein tertiary structures obtained from the BRENDA enzyme database.

4.2 Model Architecture and Training Procedure

The model architecture consists of K blocks made up of a GCN [7] convolution followed by a graph pooling. A Readout layer is applied after each block using a concatenation of the average and the maximum of vertices' features matrix $X^{(l)}$. At the end of our network, the K Readout are summed and the result is sent to a Multilayer Perceptron. The latter is composed by three fully connected layers (respectively 256, 128 and 64 for the number of hidden neurons) and a droupout of 0.5 is applied between the first two. The classification is obtained by a Softmax layer (see Fig. 1).

For the training procedure, we use Pytorch Geometric and we evaluate our neural network with a 10-fold cross validation. We repeat this procedure ten times without setting the seed. The dataset is split in three parts: 80% for the training set, 10% for the validation set and 10% for the test set.

For the hyperparameters, we use the Adam optimizer, set the dimension of node representation, the batch size and the number of epochs respectively at 128, 512 and 1000 and an early stopping is applied if the validation loss did not improved after 100 epochs. A grid search is used for the learning rate within $\left\{1e^{-3}, 1e^{-4}, 1e^{-5}\right\}$, the weight decay within $\left\{1e^{-3}, 1e^{-4}, 1e^{-5}\right\}$ and the number of blocks K in $[3, 5]$ to find the best configuration.

4.3 Ablation Studies

Scoring function. encodes the relevance of each vertex and as a direct influence on the set of selected surviving vertices by MIVSPool. We vary it using a uniform random

Score function	**MIVSPool**$_{rand}$	**MIVSPool**$_{Top-k}$	**MIVSPool**$_{SagPool}$	**MIVSPool**$_{MVPool}$
D&D	77.04 ± 0.63	77.10 ± 1.00	75.10 ± 0.76	$\mathbf{77.38 \pm 0.94}$
PROTEINS	75.15 ± 0.44	75.36 ± 0.60	75.11 ± 0.74	$\mathbf{75.62 \pm 0.47}$
NCI1	72.16 ± 0.55	$\mathbf{73.82 \pm 0.94}$	73.72 ± 0.71	72.97 ± 0.71
ENZYMES	45.80 ± 1.35	37.55 ± 1.94	38.68 ± 2.81	$\mathbf{46.80 \pm 1.53}$

Table 2: Study of MIVSPool according to the score function

Dataset	**Pooling 1**	**Pooling 2**	**Pooling 3**	**Pooling 4**	**Pooling 5**
D&D	4.1 ± 0.6	3.6 ± 0.6	3.2 ± 0.6	-	-
PROTEINS	3.1 ± 0.7	2.7 ± 0.6	2.3 ± 0.5	-	-
NCI1	3.2 ± 0.5	2.9 ± 0.5	2.5 ± 0.5	2.3 ± 0.5	2.1 ± 0.3
ENZYMES	3.2 ± 0.6	2.7 ± 0.6	2.4 ± 0.5	-	-

Table 3: Averaged number of iteration of MIVS for each pooling step.

variable $\mathcal{U}([0,1])$ like in Meer [9] (MIVSPool$_{rand}$), a trainable normalized projection vector on the features $X^{(l)}$ like in [5] (MIVSPool$_{Top-k}$), a self-attention mechanism thanks to a graph convolution like in SagPool [8] (MIVSPool$_{SagPool}$) and finally a trainable multi-view system across structure and features graph information like in MVPool [15] (MIVSPool$_{MVPool}$). For trainable variations, Equation (11) allows to propagate the training to the next convolution and, therefore, our pooling method is end-to-end trainable. The results reported Table 2 show that the choice of the score function has a minor influence on the accuracy. Note that, since Top-*k* and SagPool trainable score functions don't consider structural information, they consequently perform poorly on ENZYMES. However, the use of the MVPool trainable score function allows to obtain the best accuracy on three datasets with comparable standard deviations to other heuristics. In the rest of our article, we choose to take MIVSPool$_{MVPool}$ as a reference that we denote MIVSPool.

Average number of iterations required by MIVSPool for each pooling step. Table 3 presents the mean number of iterations for each dataset and each pooling step computed over 10 epochs. Note that the number of pooling steps is determined using K-folds for each dataset. Despite an important difference between graph sizes among datasets (Table 1), we note that the number of iterations is comparable between them and less than 5. Knowing that an iteration of MIVS on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ requires about $|\mathcal{V}|d_{max}$ computations, where d_{max} is the maximum degree of $\mathcal{G}$, the computation of MIVS on one of the graphs of our three datasets is bounded by $5|\mathcal{V}|d_{max}$. This complexity is less than the one needed by the Kron transformation (Section 2.1).

4.4 Comparison of MIVSPool according to other methods

We compare our method to three state-of-art methods: gPool [5], SagPool [8] and MVPool-SL [15]. Results in Table 4 show that MIVSPool$_{comp.}$ and MIVSPool obtain the highest or second highest accuracies on D&D, PROTEINS and ENZYMES. This point confirms the efficiency of MIVSPool and the fact that some configurations (highly connected graphs) may induce low decimation rate by MIVSPool which slightly decreases the accuracy. For NCI1, the highest accuracy is obtained by MVPool-SL [15], MIVSPool$_{comp.}$ being ranked second. Let us note that the mean vertex's degree in this dataset is 2.15 ± 0.11 with many graphs being almost linear. On such simplified topology considering two hops has done by MVPool-SL allows to recover the structure of the reduced graph.

Dataset	**gPool [5]**	**SagPool [8]**	**MVPool-SL [15]**	**MIVSPool**	**MIVSPool$_{comp.}$**
D&D	75.76 ± 0.82	75.92 ± 0.92	77.26 ± 0.37	77.38 ± 0.94	$\mathbf{77.88 \pm 0.73}$
PROTEINS	73.49 ± 1.44	74.30 ± 0.62	75.04 ± 0.67	75.62 ± 0.47	$\mathbf{75.81 \pm 0.75}$
NCI1	71.66 ± 1.04	72.86 ± 0.68	$\mathbf{74.79 \pm 0.58}$	72.97 ± 0.71	73.44 ± 0.68
ENZYMES	36.93 ± 2.45	35.30 ± 1.80	39.45 ± 2.57	$\mathbf{46.80 \pm 1.53}$	45.60 ± 2.37

Table 4: Comparison of MIVSPool according to other hierarchical methods. Highest and second highest accuracies are respectively in **bold** and blue.

5 Conclusion

Our graph pooling method MIVSPool is based on a selection of surviving vertices thanks to a Maximal Independent Vertex Set (MIVS) and an assignment of non-surviving vertices to surviving ones. Unlike state-of-art methods, our method allows to preserve the totality of graph information during its reduction.

Acknowledgements: The work was performed using computing resources of CRIANN (Normandy, France).

References

1. Balcilar, M., Renton, G., Héroux, P., Gaüzère, B., Adam, S., Honeine, P.: Analyzing the expressive power of graph neural networks in a spectral perspective. In: International Conference on Learning Representations (2021)
2. Bianchi, F.M., Grattarola, D., Livi, L., Alippi, C.: Hierarchical representation learning in graph neural networks with node decimation pooling. IEEE Trans. Neural Networks Learn. Syst. **33**(5), 2195–2207 (2022)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), 47–56 (2005)
4. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Journal of molecular biology **330**(4), 771–783 (2003)
5. Gao, H., Ji, S.: Graph u-nets. In: international conference on machine learning. pp. 2083–2092. PMLR (2019)
6. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30**, 1024–1034 (2017)
7. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
8. Lee, J., Lee, I., Kang, J.: Self-attention graph pooling. In: International conference on machine learning. pp. 3734–3743. PMLR (2019)
9. Meer, P.: Stochastic image pyramids. Computer Vision, Graphics, and Image Processing **45**(3), 269–294 (1989)
10. Nouranizadeh, A., Matinkia, M., Rahmati, M., Safabakhsh, R.: Maximum entropy weighted independent set pooling for graph neural networks. ArXiv **abs/2107.01410** (2021)
11. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
12. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**(3), 347–375 (2008)
13. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. Advances in neural information processing systems **31**, 4805–4815 (2018)
14. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. Proceedings of the AAAI Conference on Artificial Intelligence **32**(1), 4438–4445 (2018)
15. Zhang, Z., Bu, J., Ester, M., Zhang, J., Li, Z., Yao, C., Huifen, D., Yu, Z., Wang, C.: Hierarchical multi-view graph pooling with structure learning. IEEE Transactions on Knowledge and Data Engineering (2021), in Press