Quasi-CliquePool: Hierarchical Graph Pooling for Graph Classification

Waqar Ali
Ca' Foscari University of Venice
Venice, Italy
ZHAW School of Engineering
Winterthur, Switzerland
waqar.ali@unive.it

Sebastiano Vascon
Ca' Foscari University of Venice
European Center for Living Technology
Venice, Italy
sebastiano.vascon@unive.it

Thilo Stadelmann
ZHAW School of Engineering
Winterthur, Switzerland
stdm@zhaw.ch

Marcello Pelillo
Ca' Foscari University of Venice
Venice, Italy
pelillo@unive.it

ABSTRACT

Graph Neural Networks (GNNs) have revolutionized graph learning through efficiently learned node embeddings and achieved promising results in various graph-related tasks such as node and graph classification. Within GNNs, a pooling operation reduces the size of the input graph by grouping nodes that share commonalities intending to generate more robust and expressive latent representations. For this reason, pooling is a critical operation that significantly affects downstream tasks. Existing global pooling methods mostly use readout functions like *max* or *sum* to perform the pooling operations, but these methods neglect the hierarchical information of graphs. Clique-based hierarchical pooling methods have recently been developed to overcome global pooling issues. Such clique pooling methods perform a hard partition between nodes, which destroys the topological structural relationship of nodes, assuming that a node should belong to a single cluster. However, overlapping clusters widely exist in many real-world networks since a node can belong to more than one cluster. Here we introduce a new hierarchical graph pooling method to address this issue. Our pooling method, named *Quasi-CliquePool*, builds on the concept of a quasi-clique, which generalizes the notion of cliques to extract dense incomplete subgraphs of a graph. We also introduce a *soft peel-off* strategy to find the overlapping cluster nodes to keep the topological structural relationship of nodes. For a fair comparison, we follow the same procedure and training settings used by state-of-the-art pooling techniques. Our experiments demonstrate that combining the Quasi-Clique Pool with existing GNN architectures yields an average improvement of 2% accuracy on four out of six graph classification benchmarks compared to other existing pooling methods.

SAC'23, March 27 – March 31, 2023,, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/xx.xxx/xxx_x

KEYWORDS

graph neural networks; clique relaxation; quasi-clique; graph pooling; hierarchical pooling

ACM Reference Format:
Waqar Ali, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. 2023. Quasi-CliquePool: Hierarchical Graph Pooling for Graph Classification. In *Proceedings of ACM SAC Conference (SAC'23).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

In recent years, Convolutional Neural Networks (CNNs) have demonstrated outstanding performance in various challenging tasks in the fields of image processing, video processing, natural language interpretation, and beyond [12, 24]. These tasks typically represent data in euclidean space, whereas a large amount of data exists in non-euclidean domains, such as chemical molecules, biological and social networks, which can usually be represented as graphs [6]. Therefore, attempts have been made to successfully generalize CNN models to operate on graph data, leading to Graph Neural Networks (GNNs). GNNs have been implemented for various kinds of graphs and have achieved state-of-the-art performance for many graph-related tasks, such as classifying nodes, classifying graphs, and predicting links [42]. These findings demonstrate that GNNs are effective at node-level and graph-level representations. On the other hand, pooling approaches are demonstrated to be efficient and effective in many natural language processing [17] and image-related tasks such as text and image classification [19]. It is thus natural to investigate these techniques also for graph data [18]. The researchers generalize the CNN pooling methods on graphs to reduce the size of nodes for graph-level prediction. For example, studies have extended the global average or sum pooling operations to graph models by averaging or summing all node features [31]. But, such pooling methods are not able to capture the hierarchical graph structure and may lose important features [38]. Several advanced graph pooling techniques, like DiffPool [38], Topk-Pool [10], and SortPool [40], have been developed to overcome the shortcomings of global pooling and have achieved promising results on graph classification tasks. Furthermore, in [1, 22], the authors proposed topology-based clique and k-plex hierarchical pooling methods for graph classification. However, the pooling techniques

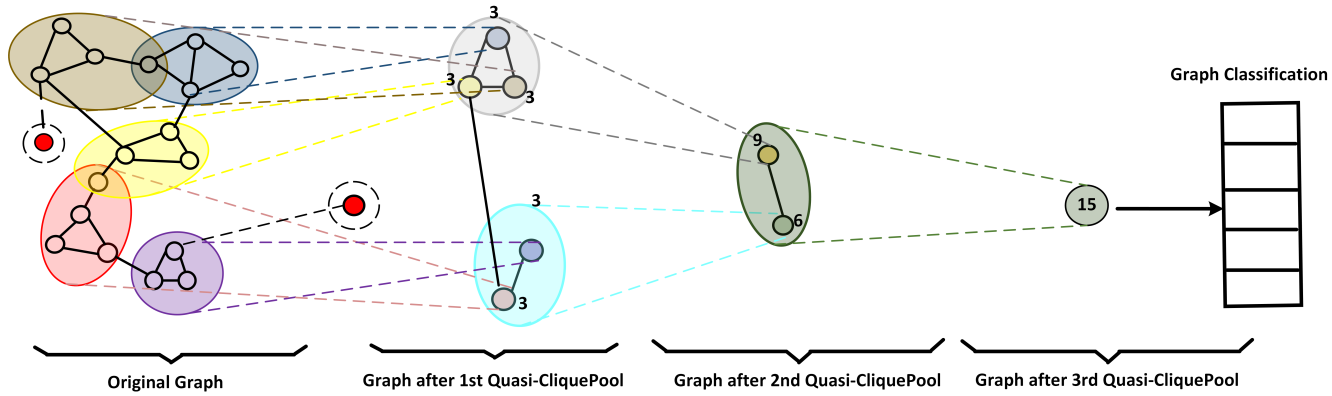

Figure 1: An example of the Quasi-CliquePool method for a graph. We run a GNN model at each hierarchical layer to obtain embeddings of nodes. After that, apply a Quasi-CliquePool on learned embeddings to get the maximal and quasi cliques and transfer each clique into new nodes in the coarsened graph $G\prime$. This process is repeated for n layers, and the final output representation is used to classify the graph. The red nodes are isolated nodes and are removed during the pooling operations. Numbers in bold font represent the number of agglomerated nodes (the highly participating ones) from the previous layer. The nodes are connected in coarsened graph based on node and edge-sharing among cliques.

mentioned above have room for improvement. For instance, the DiffPool method produces a dense adjacency matrix due to its differentiable nature. It requires hyper parameterization in the form of a prior on the number of clusters or nodes allowed to be pruned. Top-k pooling introduced a new gPool method to overcome this issue [10]. Based on the scalar projection values of the nodes, the gPool method selects the top-k nodes to reduce the size of the graph. But, this method ignores the topological structure of the graph. In [22], the authors proposed a clique pooling method for graph classification using topological information. Still, clique pooling is much more restricted and less flexible than k-plex pooling because it is limited to hard graph partitions [1]; hence, k-plex pooling has achieved good results compared to clique pooling. However, the k-plexPool depends on the k number of adjacent nodes. The method proposed in this paper is close to clique and k-plex pooling methods.

Our work links pooling operators and two graph theoretical concepts: clique and quasi-clique. The former performs a hard partitioning between nodes, where each node is connected up to one cluster. The latter ones provide flexible partitioning for a clique, which relaxes the definition of a clique to a quasi-clique to extract dense, incomplete subgraphs within a large graph. In this paper, we propose *Quasi-CliquePool* (cf. Figure. 1), a novel pooling technique to learn a hierarchical representation of a graph to address the limitations mentioned above. The proposed method uses the Replicator Dynamics (RD) algorithm to extract a subset of nodes (maximal clique) iteratively. The RD is a dynamical system that, at convergence, provides the likelihood of participation of each node into a cluster; the higher the value, the central the node.

We introduce a new soft peel-off strategy to find the low-partici pating nodes in a converged RD. Such nodes, being not central, lie on the border of the cluster. Hence they are good candidates for being the link with other clusters in the graph. Those nodes are allowed to get extracted again with other clusters, while the central ones belong only to one cluster. Having nodes that belong to multiple clusters inevitably extends the maximal clique concept to a quasi-clique due to missing edges. Nodes that remain isolated during the pooling operations are removed from the graph. Details are discussed in Section 3. We make a coarsened graph based on extracted cliques in the last step. Overall, this research makes the following contributions:

- We propose a novel graph pooling method, Quasi-CliquePool, based on the concept of quasi-clique. The proposed graph pooling method can be integrated into various GNN architectures.
- This study introduces a new soft peel-off strategy to find the overlapping cluster nodes of a given graph during the pooling procedure.
- We conduct comprehensive experiments and show that Quasi-CliquePool improves an average accuracy by 2% in four out of six graph classification benchmarks compared to state-of-the-art pooling methods.

2 RELATED WORK

This section briefly explains the multiple GNN architectures and graph pooling methods for graph classifications.

2.1 Graph Neural Networks

GNN models have drawn considerable attention due to their excellent performance on various tasks in the graph learning representation domain. Recently, several GNN architectures have been developed, including architectures inspired by CNNs [32], recursive graph networks [30], recurrent graph networks [21], and line graph neural networks [5]. Gilmer et al. [11] proposed "a neural message passing" framework for graph data, and most of the above approaches fit within this framework. In this message-passing framework, the GNN model computed the node representations directly from their neighbour nodes' features using a differentiable aggregation and propagation function. In [14], the authors present a

comprehensive review of recent advancements in this domain, and [4] makes connections to spectral graph convolutions.

2.2 Graph Pooling Methods

The pooling methods reduce the graph size using node dropping or node pruning for graph classification. There are three categories of graph pooling: topology, global and hierarchical pooling [18].

2.2.1 Topology based pooling. Earlier works used only graph coarsening methods without neural networks. For example, spectral clustering methods achieve coarsened graphs using eigendecomposition [34]. However, the eigendecomposition procedure was not good in terms of time complexity. Dhillon et al. [7] first time proposed a Graclus method to extract the clusters of given graphs without eigenvectors. Graclus method used the concept of mathematical equivalence between a general weighted kernel k-means objective and a general spectral clustering objective. This method improves the various weighted graph clustering objectives like normalized cut, ratio cut, and ratio association criteria. Even in recent GNN models [2, 28], Graclus is implemented as a pooling module.

2.2.2 Global pooling. Global pooling methods use neural networks or summation functions to pool all the node representations in each layer. Gilmer et al. [11] introduced a message-passing scheme based on a general framework for graph classification and obtained the entire graph classification using the Set2Set model. In [40], the authors proposed the SortPooling method to keep much more node information and learn from the global graph topology. This method sorts the nodes embeddings according to the graph structural roles and then feed these sorted embedding to the next layers.

2.2.3 Hierarchical pooling. Global pooling is the most effective way to reduce the size of the graph. However, these methods ignore the hierarchical graph information, which is important for capturing the structural information of graphs. The principal goal of hierarchical pooling approaches is to build a technique that uses graph topology or node feature information to learn the node representation hierarchically. In this regard, Ying et al. [38] proposed the first hierarchical differentiable pooling (DiffPool) method for classifying graphs. This method can be used with various GNN architectures in an end-to-end fashion. It used the learning assignment matrix that contains the probability values of nodes in layer L and then assigned these values to clusters in the next layer $L+1$. Due to its differentiable essence, its application produces dense adjacency matrices. In [10], the authors developed the Top-k pooling to overcome this issue by learning a project vector. But, this method ignores the topological structure of the graph. Enxhell et al. [22] introduced a clique-based hierarchical pooling method for graph classification. Clique pooling is much more restrictive because it is limited to hard graph partitions. In [1], the authors improved the simple clique pooling method and proposed a k-plex pool method for graph classification.

To further improve pooling methods, this study proposes a novel Quasi-CliquePool method that can use topological information to yield hierarchical representations. Table 1 summarizes the current graph pooling methods based on their four desirable clustering properties: 1) Hierarchical pooling—Global pooling methods ignore the hierarchical structure information in the graphs during pooling operation however, hierarchical pooling methods extract the hierarchical information. Our quasi-pooling can extract hierarchical information from graphs and can be combined with various GNN architectures, 2) Adaptive—we can distinguish pooling methods based on the k number of nodes of the pooled graph. The k-plex [1] is a fixed pooling method because it depends on the apriori k number of adjacent nodes. Our pooling method is adaptive because it is not dependent on any k number of adjacent nodes, 3) Topology pooling—in terms of graph clustering, topology structure-based aggregation is crucial for pooling operations. DiffPool [38] and Top-k [10] only used the node features to perform pooling and ignored the topological structure. The proposed method uses topological structure information to perform pooling operations, and aggregates node features using element-wise sum or max functions, and 4) Overlapping nodes—one node may belong to multiple clusters. It can be observed from Table 1 that the proposed pooling method is different from other partitioning-based graph coarsening approaches [1, 22] because Quasi-CliquePool extracts overlapping nodes during pooling operations to preserve the topological structure. Clique pooling, on the other hand, forces a split between two nodes, which destroys the topological relationship between the two nodes. The k-plex pooling finds the overlapping nodes, but it depends on the k-fixed number of adjacent nodes.

Table 1: Related work in terms of four desirable graph pooling properties outlined in Section 2. Methods are divided into hierarchical (H), adaptive (F), topology pooling (TP), and overlapping nodes (ON).

Methods	H	A	TP	ON
Graclus [7]	✘	✔	✔	✘
TopK-Pool [38]	✔	✔	✘	✘
SAGPool [18]	✔	✔	✔	✘
DiffPool [10]	✔	✘	✘	✔
SortPool [40]	✘	✔	✔	✘
CliquePool [22]	✔	✔	✔	✘
K-plexPool [1]	✔	✘	✔	✔
Quasi-CliquePool	✔	✔	✔	✔

3 QUASI-CLIQUE GRAPH POOLING

In this section, we explain the mechanism of Quasi-CliquePool and show how it is implemented in a GNN architecture for graph classification. Section 3.1 defines some useful notations. Sections 3.2 and 3.3 briefly describe the background for GNNs and RD with the quasi and maximal cliques. In section 3.4, we explain how the quasi-clique method can be used to coarsen the graph. Finally, section 3.5 explains the Quasi-CliquePool algorithm to extract the quasi-cliques and maximal cliques.

3.1 Notations

Given an undirected graph $G = (V, E)$, where $E = E(G)$ is an edge set, $V = V(G)$ is a node-set, $e(G) = |E| = m$ and $v(G) = |V| = n$ are the number of edges and nodes in G, respectively. Given an edge $e = \{u, v\}$, nodes u and v are said to be adjacent. We represent an attributed graph as a tuple (G, α, β), where $\alpha : V \rightarrow \mathbb{R}^{h_V}$ is a function that assigns a vector of features to each node and $\beta : V \times V \rightarrow \mathbb{R}^{h_E}$ is an edge feature function that assigns a feature

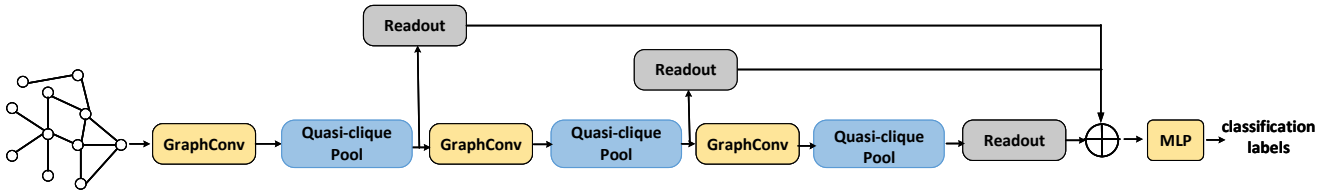

Figure 2: Hierarchical Quasi-clique pooling framework for graph classification. GraphConv is applied to obtain embeddings of graphs at each hierarchical layer. After that, a Quasi-CliquePool layer is applied to the learned embeddings to get a new graph G_r. The proposed hierarchical framework is composed of three blocks: three GraphConv layers and three quasi-clique pooling layers. The sum readout function is applied after every block. Lastly, add the embeddings of each graph layer to get the final graph representation, and a multi-layer perceptron model uses this representation to classify the original graph.

vector to each edge, and h_V and h_E represent the size of edges and nodes. An attributed graph can also be represented in matrix form (A, X), where $A \in \{0,1\}^{n\times n}$ denotes the adjacency matrix with $A_{ij} = A_{ji} = \beta(\{v_i, v_j\})$. The term $X \in \mathbb{R}^{n\times h_V}$ denotes the node feature matrix, and the rows of X are defined as $x_i = \alpha(v_i)$.

3.2 Graph Convolutional Neural Network

GNNs are a special class of neural network architectures designed to analyze graph data. These models learn a fixed-size representation of a set of graphs in a given distribution. The main component of the GNN model is the "message passing module" that is described below:

$$H^{(k)} = P(A, H^{(k-1)}; \Theta^{(k)}), \tag{1}$$

Where P represents the propagation function, which depends on three components: adjacency matrix A, node embedding H^{k-1} based on the previous message passing step, and the last term Θ are trainable parameters. There are many ways to implement the propagation function P. For example, Kipf's et al. [36] proposed a Graph Convolutional Neural Network (GCN) implement the P function using the combination of ReLU non-linearities and linear transformations:

$$H^{(k)} = P(A, H^{(k-1)}; W^{(k)}) = ReLU(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k-1)}W^{(k-1)}), \tag{2}$$

where $\tilde{A} = A + I$, $\tilde{D} = \sum_j \tilde{A}_{ij}$ and $W^{(k)} \in \mathbb{R}^{n\times n}$ is trainable weight matrix. This architecture has achieved promising performance in many challenging tasks, such as graph and node classification. Our proposed Quasi-CliquePool can be integrated into different GNN models like GCN, GraphSAGE, and GraphConv. A detailed explanation is mentioned in the experimental section.

3.3 Replicator Dynamics, Maximal and Quasi Clique

Replicator Equations (REs) are a class of dynamical systems developed to model the evolution of animal behaviour using tools and principles of game theory. The REs have recently been applied with significant success to solve the maximal clique and related problems [27]. Based on a well-known result from graph theory, this approach formulates the maximal clique problem as a standard quadratic assignment program. We introduce some notations and definitions to represent this concept formally. Initially, the data to be clustered are represented as a graph $G = (V, E)$ with no self-loop,

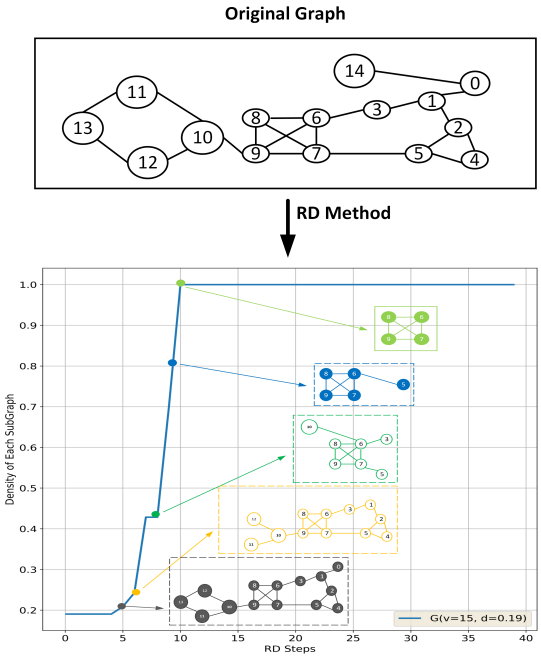

Figure 3: Illustrations of the RD method to obtain the quasi-cliques and maximal clique. Given a graph with 15 nodes and an initial density ($d = 0.19$). We apply the RD method to this graph and map the obtained quasi-cliques and maximal clique onto the density curve at each iteration of the RD. The quasi-cliques and maximal clique are shown in the dotted line and solid line boxes, respectively. It can be seen that the RD method returns the quasi-cliques at each step; however, it returns the maximal clique at convergence when it reaches density $d = 1.0$.

where V and E are sets of vertices and edges, respectively. In our case, the vertices correspond to the graph nodes, and the edges represent the neighbouring relationship between two nodes. We compute the adjacency matrix of G, which is the $n \times n$ non-negative symmetric matrix $A = (A_{uv})$ defined as follows:

$$a_{uv} = \begin{cases} 1, & if(u,v) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The degree of a vertex $u \in V$ relative to a subset of vertices C, denoted by $deg_C(u)$, is the number of vertices in C adjacent to it, that is,

$$deg_C(u) = \sum_{v \in C} a_{uv}. \tag{4}$$

When $C = V$ we obtain the standard degree notion, in which case we shall write $deg(u)$ instead of $deg_V(v)$. A subset C of vertices in G is called a clique if all its vertices are mutually adjacent. A clique is said to be maximal if it is not contained in any larger clique, while Quasi-clique is a dense incomplete subgraph of a graph that relaxes the clique constraints. We define the quasi-clique in terms of density d such that $0 < d < 1$, a subset of vertices H is called a d-quasi-clique if the edge density of the induced subgraph $G[H]$ is at least d. Note that if $d = 1$, the definition requires H to be a clique. If $d < 1$, it is possible for the subgraph to be missing some edges among its vertices. In [27], a one-to-one correspondence between stable points of the RD, local maximizers of a standard quadratic assignment problem and maximal clique is provided. It is thus sufficient to reach an equilibrium point of the RD to get a maximal clique. We used this algorithm in our implementation to extract the maximal and quasi-cliques. Figure. 3 shows the graphical representation of how the RD method obtains the maximal cliques and quasi-clique using a density curve. The discrete replicator dynamic is defined as:

$$x_i^{(t+1)} = x_i^{(t)} \frac{(Sx^{(t)})_i}{(x^{(t)})' S(x^{(t)})} \tag{5}$$

the equation. 5 (for $i = 1, ..., n$) corresponds to the discrete-time version of first-order replicator equations. The RD is a continuous optimization technique that, at convergence, provides the degree of centrality for each node in a cluster; the higher the degree, the more central the node. In [27], the authors introduced an effective strategy (Peel-off) to perform a hard partition of the given data into coherent clusters using RD, with the following steps: 1) find the most participating nodes based on a predefined threshold to obtain a cluster. (i.e., a maximal clique), 2) remove those selected nodes from the similarity graph, and 3) reiterate steps 1 and 2 on the remaining nodes until all nodes have been clustered. The peel-off strategy considers only the highly participating nodes due to the predefined threshold in the RD convergence; however, the low-participating nodes that lie on the border of the cluster might be a good candidate to link with other clusters in the graph. These nodes can be extracted again with other clusters at RD convergence, while the central ones belong only to one cluster. Hence we introduce a new soft peel-off strategy, a threshold, to find these low participating nodes in a converged RD. Figure. 4 shows that the central nodes belong only to one cluster, while the border nodes can be linked to other clusters. The run-time complexity of RD is $O(K|V|^2)$, where V is the number of vertices in the graph and K is the number of iterations. A detailed explanation of how we used the RD algorithm and soft peel-off strategy in our implementation is mentioned in section 3.5.

3.4 Graph Coarsening with Quasi-CliquePool

The proposed Quasi-CliquePool method computes the cliques $C = \{C_1, C_2, ..., C_k\}$ of the input graph (V, E, α, β), and returns a coarsened graph $(V', E', \alpha', \beta')$, such as:

$$V' = V(G') = \{v'_1, v'_2, v'_3, ..., v'_k\}, \tag{6}$$

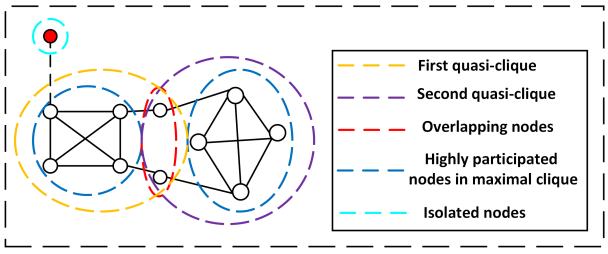

Figure 4: An example illustrates the clustering process using Quasi-CliquePool. In the first step, the RD method extracts a yellow quasi-cluster with six nodes: four highly participated and two overlapping nodes. The red one is an isolated node removed during the pooling procedure. The overlapping nodes are selected from the yellow cluster using the *soft peel-off* strategy σ. We removed only the highly participated nodes from the similarity matrix and iterated the RD method to extract the second purple cluster with overlapping nodes.

$$E' = E(G') = \{\{v'_i, v'_j\} \mid E(G[C_i, C_j]) \neq \emptyset\}, \tag{7}$$

where node v'_i represents the coarsened version of C_i and $E'\{v'_i, v'_j\}$ represents coarsened edge that exists iff there is at least one edge in original graph G connecting a node of C_i with a node of C_j. The node features function $\alpha' : V' \rightarrow \mathbb{R}^{h_V}$ aggregates the features that belong to the same clique C_i. We considered the features of maximal clique-based nodes for aggregation since these nodes are highly connected. For relabeling the nodes and edges in the coarsened graph, we defined node features in the following way:

$$\alpha'(v_i)' = \Phi(\{\alpha(v) | v \in C_i\}), \tag{8}$$

$$\beta'(\{v'_i, v'_j\}) = \psi(\{\beta(e) | e \in E(G[C_i, C_j])\}), \tag{9}$$

where ϕ and ψ represent the aggregation functions defined over multisets of feature vectors. The element-wise max or sum is a common aggregator function for node features [37]. We used element-wise max or sum aggregators for node features. Our approach is different from other partitioning-based graph coarsening methods [22] because a node may belong to multiple cliques in the proposed method. On the other hand, CliquePool [22] performs a hard partition between nodes, which destroys the topological structural relationship in the cliques[1]. Figure 2 illustrates the framework of Quasi-CliquePool. Concretely, we view a GraphConv layer followed by a Quasi-cliquePool layer as a module and name it Quasi-cliquePool GraphConv layer for convenience. The Quasi-cliquePool GraphConv layer takes a graph as an input and outputs a new pooled graph with a new feature matrix and adjacency matrix. Then, the pooled graph is fed into the next Quasi-cliquePool GraphConv layer and applies a readout function on it. In the last step, we get the final graph representation by adding the embeddings of each graph layer, and a multi-layer perceptron model is applied to this representation to classify the graph.

3.5 Quasi-CliquePool Algorithm

In this section, we propose a Quasi-CliquePool algorithm for graph classification, whose pseudocode is shown in Algorithm 1, that extracts the quasi-cliques and maximal cliques of a given graph.

[1]We do not consider edge attributes in this work.

Algorithm 1 performs two main tasks: 1) partitions the graph and assigns the nodes of the original graph G to k different cliques, and 2) each clique is transformed into a new node in the coarsened graph G'. The core of our method is inspired by the original RD algorithm for the maximal clique of Pelillo et al. [27]. Algorithm 1 receives a graph $A \in \{0,1\}^{n\times n}$ and $X \in \mathbb{R}^{n\times h_V}$ as input and returns the list of all possible quasi-cliques C. In the first step, we used the Euclidean pairwise distance method, and the gaussian kernel [39] function to build the similarity matrix of the given graph (equation.10). The kernel function basically tells the model how similar two data points are (x_i, x_j). The affinity between a pair of points can be defined as

$$S_{ij} = \exp\left(\frac{-d^2(x_i, x_j)}{\rho_i \rho_j}\right), \tag{10}$$

where $d(x_i, x_j)$ is the euclidean distance between the vectors x_i and x_j and ρ_i ρ_j are the local scaling parameters computed with [39]. In the next step, the RD method uses this similarity matrix as an input and provides a characteristic vector X_n that contains the probability value of the participation of each node in a cluster; the higher the value, the central the nodes as a maximal clique, and the lower the value, the border the nodes. Then a γ threshold is applied to the characteristic vector X_n to extract the highly participating nodes as a maximal clique. Here we are interested in extracting the low participating nodes that lie on the clique's border because these nodes might be good candidates for overlapping nodes with other cliques in the graph. We introduced a new soft peel-off strategy, a filter operation that finds border (overlapping) nodes using a σ threshold and assigns them again in characteristic vector x for the next convergence of the RD. So in this way, Algorithm 1 iteratively selects all the possible quasi-cliques by extracting the nodes in set U. Nodes that remain isolated during the pooling operations are removed from the graph. Moreover, the aggregation procedure of the node features (equation. 8) contributes similarly to the respective clusters, and element-wise sum or average readout functions are used to aggregate the node features in the coarsened graph[2]. In the next pooling layer, we transformed each clique into a new node and connected two cliques (equation. 9) if they have a common edge or node in the original graph G. We illustrate our proposed Quasi-CliquePool in Figure. 1, where we performed quasi-clique pooling on a graph with 19 nodes and obtained a new graph G' at the first layer of Quasi-CliquePool with 5 nodes.

4 RESULTS AND DISCUSSION

In this section, we evaluate the superiority and effectiveness of Quasi-CliquePool against several state-of-the-art graph classification approaches. Section 4.1 briefly describes the datasets used for experiments. Section 4.2 explains the baseline methods used to compare the results and configuration of Quasi-CliquePool and baseline methods. In section 4.3, we compare Quasi-CliquePool results with state-of-the-art graph classification approaches. Finally, section 4.4 presents the ablation study.

[2]We only considered the features of highly connected nodes in a quasi-clique for feature aggregation. For this purpose, we used the Bron Kerbosch approach to aggregate the features of maximal clique-based nodes.

Algorithm 1 Quasi-CliquePool

Input: *Given a graph G as* $A \in \{0,1\}^{n\times n}$ *and* $X \in \mathbb{R}^{n\times h_V}$
Output: *List of all possible Quasi − Cliques C of G*

```
C ← ∅
S ← build a similarity matrix (equation.10)
U ← V(G)
while U ≠ 0 do
    X_n ← RD (S)            ▷ characteristics vector X_n (equation. 5)
    C_i ← filter(X_n, γ)            ▷ extracts the quasi-clique
    L ← filter(C_i, σ)      ▷ selects the low participated nodes
quasi-clique (overlapping nodes) from (C_i)
    C ← Append (C_i)
    C_i ← C_i \ L           ▷ remove the selected lowest value nodes
from C_i
    Remove C_i from the similarity matrix S
end while
C = {C_1, ..., C_k}
```

4.1 Dataset Setup

To verify the performance of Quasi-CliquePool in learning complex hierarchical graph structures in different domains, it tested on a variety of large benchmarks that are commonly used in graph classification tasks [15]. This study used bio-informatics datasets including DD [9], Protein and Enzymes [3, 8], and the molecule datasets Mutag [29], NCI-1, and NCI-109 [33]. All datasets are retrieved from the TU-Dortmund collection [25]. Table 3 shows the statistics of the datasets, most of the datasets are relatively large-scale with different sizes of graphs and hence suitable for evaluating deep graph models.

4.2 Baselines and Experimental Settings

To compare the performance of graph classification, we consider GNNs-based baselines combined with different state-of-the-art pooling methods. The next section briefly describes these baseline methods with experimental settings.

4.2.1 Graph Neural Network Methods: This research work considers three GNN architectures to test the proposed Quasi-CliquePool method. (1) GCN [16] is a convolutional neural network that learns node representations by aggregating and propagating information from neighbours. (2) GraphSage [13] introduces the inductive framework, which calculates node embedding by aggregating and sampling features from local neighbours. (3) GraphConve [26] proposed k-dimensional GNNs that can take high-order graph structures and are useful in analysing social networks and molecule graphs.

4.2.2 Competitors: This study compares Quasi-CliquePool with five state-of-the-art hierarchical pooling techniques: Top-k pool [38], SAGEpool[18], DiffPool [10], clique pooling [22], and k-plex pool[1] and two global pooling methods: Sort pooling[35] and Graclus[7]. For all baselines and quasi-clique poling, we employed the same hyperparameter search strategy. The hyperparameters are summarized in Table 4.

For all the pooling and GNN baselines, we consider the accuracy scores reported by the original authors. In cases where baseline

Table 2: Test accuracy on the classification of molecules and bio-informatics benchmarks. The bold and underlined text highlight the highest and second-highest accuracy scores, respectively.

Classification	Baselines	ENZYMES	PROTEINS	D&D	NCI-1	NCI-109	MUTAG
GNNs	GCN [16]	28.68%	70.01%	71.42%	70.24%	68.33%	72.30%
	GraphSAGE [13]	31.73%	71.37%	71.70%	73.36%	72.30%	74.08%
	GraphConv [26]	33.71%	72.43%	72.31%	74.70%	73.22%	77.97%
Pooling	Graclus	28.51%	71.35%	72.45%	74.25%	72.32%	76.64%
	TopK-Pool [38]	31.64%	<u>77.25%</u>	**82.43%**	73.30%	72.30%	74.75%
	SAGPool [18]	32.68%	70.04%	76.19%	74.18%	<u>74.04%</u>	77.72%
	DiffPool [10]	**62.53%**	76.25%	<u>80.64%</u>	76.40%	74.29%	81.09%
	SortPool [40]	-	75.54%	79.37%	74.48%	72.31%	<u>84.12%</u>
	CliquePool [22]	42.17%	73.86%	74.88%	78.83%	-	-
	K-plexPool [1]	43.33%	75.92%	77.76%	<u>79.01%</u>	-	-
Proposed	Quasi-CliquePool	<u>45.01%</u>	**78.68%**	75.30%	**80.11%**	**76.30%**	**84.88%**

Table 3: Statistics of datasets. G_{avg}, V_{avg}, E_{avg} and C_{avg} denote the average number of graphs, nodes, edges, and classes, respectively

Datasets	G_{avg}	V_{avg}	E_{avg}	C_{avg}
ENZYMES [8]	600	32.63	62.14	6
PROTEINS [3]	1,113	39.06	72.82	2
D&D [9]	1,178	284.32	715.66	2
NCI-1 [33]	4,110	29.87	32.30	2
NCI-109 [33]	4,127	29.68	32.13	2
MUTAG [29]	188	17.93	19.79	2

techniques did not publish require classification scores, we used the original authors' code (if available) with the same hyperparameters setting mentioned in the original papers. In our experiments, we used the GraphConv architecture for Quasi-CliquePool, since we achieved effective and superior performance with this architecture compared to the standard graph convolutional model. We used the Graphconv "add" variant, and after each Graphconv layer, we added a Quasi-CliquePool layer. A global readout function is applied after every layer of Quasi-CliquePool with dropout (ratio 0.5). A total of three Quasi-CliquePool layers are used for the datasets. Next, the ReLu activation function is applied after every convolutional layer. In the last, a softmax function is applied to classify the graph. We randomly split each dataset into three parts: 80 percent for training, 10 percent for the validation set, and 10 percent for the testing set. We repeated this random splitting process 10 times to get more stable performance and reported the average performance. We used PyTorch to implement the Quasi-CliquePool and the Adam optimizer to optimize the model. Table 4 shows the hyperparameter list. For all GNNs and pooling baselines, we used the official PyTorch published code. [3]

4.3 Performance on Graph Classification

Table 2 demonstrates the test accuracy of Quasi-CliquePool on bio-informatics and molecules benchmarks. To summarize the results, we have the following observations:

[3] $https://github.com/pyg-team/pytorch\_geometric/tree/master/benchmark/kernel$

Table 4: The hyper-parameters setting. We applied three graph convolutional layers and three quasi-clique pooling layers. The pooling ratio is used only for Topk-Pool, SAG-Pool, and DiffPool.

Model	Hyper-Parameters	Values
All	GNNs	GCN, GraphSAGE, GraphConv
	layers	2, 3
	learning rate	1e-2, 1e-3, 1e-4
	weight decay	1e-2, 1e-3, 1e-4, 1e-5
	TopK-Pool, SAG-Pool	0.8
DiffPool	pooling ratio	0.25
RD	ϵ, γ	2.0e-4, 1.0e-7
Quasi-CliquePool	σ	1.876e-03

- First, we can observe from the results that Quasi-CliquePool consistently outperforms other baselines in most datasets. For example, in the molecule datasets, quasi-clique pooling achieves 1.10% improvement compared to the K-plex best baseline in NCI-1, which is 10.13% improvement over a Graph Convolutional Network (GCN) with no hierarchical pooling mechanism. Quasi-CliquePool also achieves 2.76% improvement over the k-plexPool in the Proteins dataset, and the overall improvement is 8.67% over the GCN model.
- In most of the datasets, our quasi-clique pooling method obtains better performance than both hierarchical and global pooling methods. Quasi-CliquePool almost achieves 2.5% overall improvement over hierarchical baselines, including DiffPool, clique, k-plex, and Topk-Pool in bio-informatics datasets. At the same time, the quasi-clique method almost achieves 8% improvement over global pooling approaches (Graclus and Sortpooling).
- Being consistent with existing studies' findings [23, 41], we can see from Table 2 that GNN architectures without pooling modules are not able to achieve promising results because they ignore hierarchical graph information while summarizing the node representations globally. So this also proves that

GNNs need a graph pooling layer for graph classification tasks.

- We also note that the hierarchical-based pooling methods achieved relatively better results than global methods, which further demonstrates the effectiveness of the hierarchical pooling operations. Both the SAGPool and Top-k pooling methods perform poorly on the ENZYMES dataset. The possible reason may be limited training examples per class, resulting in overfitting in GNN. However, DiffPool achieves superior performance in the ENZYMES dataset, and the proposed Quasi-CliquePool achieves the second-best accuracy in this dataset. In addition, the Top-Kpool obtains the best performance on the D&D dataset, and DiffPool obtains the second-best performance. The quasi-clique pooling method performs badly on the D&D dataset because it has very large, noisy, and sparse graphs.
- In comparison to the K-Plex and clique pooling methods, our method achieves the best performance on all six datasets, as shown in Table2. Such observations demonstrate the overlapping nodes information in graphs is useful for graph pooling. And overall, our Quasi-CliquePool performs best on four out of six datasets.

4.4 Ablation Study

This section presents the ablation study to verify the performance of the proposed model by varying the sensitivity of several key hyper-parameters. Next, we integrate our pooling method into various GNN architectures to investigate its effect. We also investigate the performance of quasi-clique pooing with different readout functions.

4.4.1 Quasi-CliquePool and Graph Neural Network Architectures. As previously mentioned, the proposed Quasi-CliquePool can integrate into various GNNs architectures. We integrate Quasi-Clique Pool into the three most widely used graph convolutional models, including GCN, GraphSAGE, and GraphConv. These models test on three datasets (Protein, Enzymes, NCI-109), which cover large and small graph datasets with multiple classes. Figure. 5 shows the performance of the three Quasi-CliquePool variants. It can see the Quasi-CliquePool_GraphConv achieves the highest accuracy on all three datasets, specifically on the Protein dataset. One can also see that the performance of Quasi-CliquePool_GraphConv on the Enzymes dataset is also better than other variants, so it shows our proposed model can get good results on multi-classes datasets.

4.4.2 Hyper-Parameter Analysis. This section further investigates the sensitivity of some important hyper-parameters on Quasi-Clique Pool. In detail, we investigate how the GNN layers L and graph representation dimension d affect graph classification results. As shown in Figure. 6, Quasi-CliquePool obtains the highest accuracy when setting k = 3 and d = 64, respectively. It can be observed that the accuracy presents a slight increase trend with the dimension d increasing in both datasets. This is because the higher dimensional representation space makes classification tasks easy. One can also see that when we increase the neural network layers, the accuracy is also increasing, but too large layers L will hurt the model's performance due to over-smoothing [20].

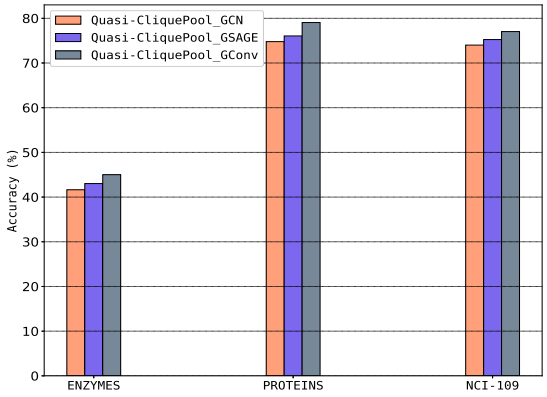

Figure 5: Quasi-CliquePool performance with different GNN

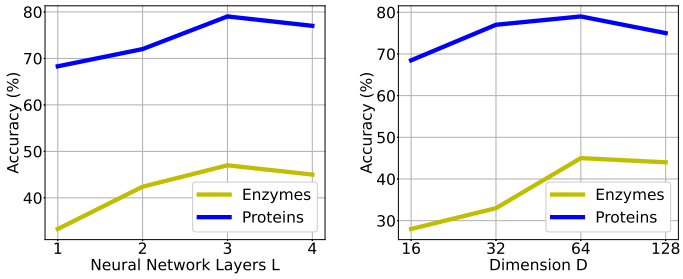

Figure 6: Test accuracy curves on Protein and Enzymes with different values of L and d

4.4.3 Readout Functions. To investigate how the different readout functions affect the performance, we used three readout functions, *sum, avg, and max* to aggregate node features, which are denoted as Quasi-CliquePool-sum, Quasi-CliquePool-avg, and Quasi-CliquePool-max, respectively. These readout-based quasi-clique models evaluate on three datasets (Protein, Enzymes, NCI-109). As shown in Table 5, Quasi-CliquePool-Max and Quasi-CliquePool-Avg achieve better performance than Quasi-CliquePool-Sum. Quasi-CliquePool-max obtains the highest accuracy than Quasi-CliquePool-Avg, which is consistent with existing research work [1, 22]. This observation highlights the significance of readout functions in aggregating nodes in pooling operations.

Table 5: Quasi-CliquePool performance with various readout functions.

Readout Functions	ENZYMES	PROTEINS	NCI-109
Quasi-CliquePool-Sum	43.20%	75.89%	78.15%
Quasi-CliquePool-Max	**45.01**%	**79.05**%	**80.11**%
Quasi-CliquePool-Avg	44.01%	77.30%	77.30%

5 CONCLUSION

In this research work, we designed a novel graph pooling technique, Quasi-CliquePool, for graph classification using the RD algorithm. The proposed Quasi-CliquePool method has the following specifications: exploits the topological structure of the graph, extracts the complex hierarchical structure of graphs, does not require a-priori knowledge of the hierarchy, and can be integrated into several GNN architectures. This study also introduced a soft peel-off strategy to find the overlapping nodes of the graph in the clustering procedure. To demonstrate the superiority of Quasi-CliquePool for graph classification, it tested on six datasets from two domains (molecules and bio-informatics). The proposed Quasi-CliquePool method obtained the best results on four of them, which demonstrates our model's effectiveness.

REFERENCES

[1] Davide Bacciu, Alessio Conte, Roberto Grossi, Francesco Landolfi, and Andrea Marino. 2021. K-plex cover pooling for graph neural networks. *Data Mining and Knowledge Discovery* 35, 5 (2021), 2200–2220.

[2] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. 2020. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*. PMLR, 874–883.

[3] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[4] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[5] Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. 2021. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* 29 (2016).

[7] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence* 29, 11 (2007), 1944–1957.

[8] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.

[9] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. 2013. Scalable kernels for graphs with continuous attributes. *Advances in neural information processing systems* 26 (2013).

[10] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *international conference on machine learning*. PMLR, 2083–2092.

[11] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.

[12] Arousha Haghighian Roudsari, Jafar Afshar, Wookey Lee, and Suan Lee. 2022. PatentNet: multi-label classification of patent documents using deep learning based language understanding. *Scientometrics* 127, 1 (2022), 207–231.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[15] Kristian Kersting, Nils M Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. 2016. Benchmark data sets for graph kernels. (2016).

[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[17] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. 2017. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 364–371.

[18] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *International conference on machine learning*. PMLR, 3734–3743.

[19] Xinyu Lei, Hongguang Pan, and Xiangdong Huang. 2019. A dilated CNN model for image classification. *IEEE Access* 7 (2019), 124087–124095.

[20] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9267–9276.

[21] Y Li, D Tarlow, M Brockschmidt, and R Zemel. 2016. Gated graph sequence neural networks In: International Conference on Learning Representations. *San Juan* (2016).

[22] Enxhell Luzhnica, Ben Day, and Pietro Lio. 2019. Clique pooling for graph classification. *arXiv preprint arXiv:1904.00374* (2019).

[23] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 723–731.

[24] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2021. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* (2021).

[25] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663* (2020).

[26] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4602–4609.

[27] Marcello Pelillo and Andrea Torsello. 2006. Payoff-monotonic game dynamics and the maximum clique problem. *Neural Computation* 18, 5 (2006), 1215–1258.

[28] Sungmin Rhee, Seokjun Seo, and Sun Kim. 2017. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *arXiv preprint arXiv:1711.05859* (2017).

[29] Kaspar Riesen and Horst Bunke. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 287–297.

[30] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[31] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3693–3702.

[32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *stat* 1050 (2017), 20.

[33] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14, 3 (2008), 347–375.

[34] Xiang Wang, Buyue Qian, and Ian Davidson. 2014. On constrained spectral clustering and its applications. *Data Mining and Knowledge Discovery* 28, 1 (2014), 1–30.

[35] Zhengyang Wang and Shuiwang Ji. 2020. Second-order pooling for graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[36] Max Welling and Thomas N Kipf. 2016. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[38] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems* 31 (2018).

[39] Lihi Zelnik-Manor and Pietro Perona. 2004. Self-tuning spectral clustering. *Advances in neural information processing systems* 17 (2004).

[40] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[41] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. 2019. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954* (2019).

[42] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. 2018. ANRL: attributed network representation learning via deep neural networks.. In *Ijcai*, Vol. 18. 3155–3161.