




K-plex cover pooling for graph neural networks

Davide Bacciu¹ · Alessio Conte¹ · Roberto Grossi¹ · Francesco Landolfi¹  · Andrea Marino²

Received: 2 February 2021 / Accepted: 26 June 2021
© The Author(s) 2021

Abstract

Graph pooling methods provide mechanisms for structure reduction that are intended to ease the diffusion of context between nodes further in the graph, and that typically leverage community discovery mechanisms or node and edge pruning heuristics. In this paper, we introduce a novel pooling technique which borrows from classical results in graph theory that is non-parametric and generalizes well to graphs of different nature and connectivity patterns. Our pooling method, named KPLEXPOOL, builds on the concepts of graph covers and k -plexes, i.e. pseudo-cliques where each node can miss up to k links. The experimental evaluation on benchmarks on molecular and social graph classification shows that KPLEXPOOL achieves state of the art performances against both parametric and non-parametric pooling methods in the literature, despite generating pooled graphs based solely on topological information.

Keywords Graph pooling · Graph neural network · K-plex · Graph covering · Pseudo-clique

Responsible editor: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautmann.

✉ Francesco Landolfi
francesco.landolfi@phd.unipi.it

Davide Bacciu
bacciu@di.unipi.it

Alessio Conte
conte@di.unipi.it

Roberto Grossi
grossi@di.unipi.it

Andrea Marino
andrea.marino@unifi.it

¹ Department of Computer Science, Università di Pisa, Pisa, Italy

² Department of Statistics, Computer Science, Applications “G. Parenti” (DISIA), Università degli Studi di Firenze, Florence, Italy

1 Introduction

Graph Neural Networks (GNNs) allow for the adaptive processing of topology-varying structures representing complex data which comprises atomic information entities (the nodes) and their relationships (the edges).

The processing of graphs by neural networks typically leverages on message passing between neighboring nodes (Bacciu et al. 2020) to collect and exchange information on the context of nodes. Such process is made effective and efficient, also on cyclic structures, by feedforward neural layers with node-level weight-sharing, an approach popularized under the term graph convolutions (Kipf and Welling 2017), but previously known as contextual structure processing (Micheli 2009; Bacciu et al. 2018).

Graph pooling methods provide mechanisms for structure reduction that are intended to ease the diffusion of such a context between nodes farther in the graph. These methods developed from the original concept in image processing, realizing structure reduction layers that are interleaved between graph convolutions to provide a multi-resolution view of the input graph. This is intended to extract coarser and more abstract representations of the graph as we go deeper in the network, boosting information spreading among nodes. In this respect, graph pooling can help to contain model complexity, by reducing the number of convolutional layers needed to achieve full coverage of the graph, and counteract oversmoothing issues (i.e. nodes converging to very similar embeddings) by introducing structural diversity among convolutional layers (Li et al. 2018).

The definition of a robust, general and efficient subsampling mechanism that can scale to topology-varying graphs is made difficult by the irregular nature of the data and by the lack of a reference ordering of nodes between samples. Graph pooling methods in the literature are addressing the problem from a community discovery perspective, more or less explicitly, by considering node connectivity patterns (Simonovsky and Komodakis 2017; Ma et al. 2019; Wang et al. 2020; Defferrard et al. 2016) or by aggregating the nodes based on their similarity in the neural embedding space (Ying et al. 2018; Lee et al. 2019; Bianchi et al. 2020).

Our work hinges on an explicit (and novel) link between pooling operators and two consolidated graph theoretical concepts in community discovery, namely, k -plexes and graph covers. The former ones provide a flexible formalization for a community of nodes as a densely interconnected and cohesive subgraph, which relaxes the definition of a clique by allowing nodes to miss up to k links each. The latter ones relate to a soft-partition of nodes whose union covers all nodes on the original graph. We argue that both concepts are necessary to realize an effective and general graph pooling mechanism as they permit to summarize the overall community structure by taking a small but meaningful set of highly connected components, represented by the k -plexes.

We introduce KPLEXPOOL, a novel pooling method using only topological graph features, which is not parameterized nor its outcomes depend on the specific predictive task. Hence, its structure reduction can be precomputed once and reused across multiple learning model configurations. We show how KPLEXPOOL, despite being non-adaptive, provides a flexible and robust definition for node communities which can generalize well to graphs of different nature and topology, from molecular structures to social networks.

We remark that KPLEXPOOL is not just a straightforward application of k -plexes to deep learning for graphs. Rather, our pooling mechanism is the result of the careful design and integration of different community discovery and graph simplification mechanisms specifically crafted to obtain a hierarchical structure coarsening algorithm well suited to promote effective context diffusion in neural message passing. This goal is quite challenging as, for instance, a previous attempt by Luzhnica et al. (2019) clearly shows that the straightforward application of clique discovery does not yield an effective graph pooling method. We hope that our work can further stimulate the community interests in graph theory and algorithms, which have been excellent sources of inspiration for the machine learning community, such as the Weisfeiler-Lehman graph kernel (Shervashidze et al. 2011; Kriege et al. 2020; Vishwanathan et al. 2010) to name one of these achievements. The main original contributions of this paper are summarized below.

- We propose KPLEXPOOL, the first pooling mechanism based on the concepts of k -plex communities and graph covers (Sect. 2.3).
- We define a scalable algorithm to compute a hierarchy of k -plex cover decompositions that optimizes context propagation and promote diversity in convolutional layers (Sect. 2.4).
- We propose a post-processing cover heuristic for sparsifying pooled graphs in scale-free structures (Sect. 2.6).
- We provide a thorough and reproducible empirical assessment on 9 graph classification benchmarks, where KPLEXPOOL is shown to be state-of-the-art (Sect. 4).

We remark that while the notion of clique-covering has been studied since the 70s, we are not aware of any formal definition or algorithm for k -plex covering in literature. Hence, our contribution is novel in: i) the use of k -plex communities to define pooling mechanisms in neural processing systems; ii) the definition of the notion of k -plex cover; iii) the definition of an efficient algorithm to implement the concepts above.

2 K -plex cover graph pooling

In this section we present KPLEXPOOL, a novel method for graph coarsening that uses k -plexes as a pooling block. In Sects. 2.1, 2.2 we introduce some useful definitions from graph theory and deep learning that will be used throughout this paper. In Sect. 2.3, we begin by defining how k -plexes can be used to coarsen connected communities in the graph and how edges can be determined in the pooled structure. In Sect. 2.4 we describe the algorithms to efficiently compute the k -plex covers. Finally, in Sect. 2.6 we introduce a useful heuristic for sparsifying scale-free structures.

2.1 Preliminaries on graph theory

Given an undirected graph G , let $V = V(G)$ be its node set and $E = E(G)$ be its edge set, where $v(G) = |V| = n$ and $e(G) = |E| = m$ are, respectively, the number of nodes and edges in G . Given an edge $e = \{u, v\}$, nodes u and v are said to be adjacent

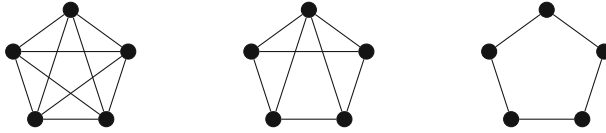


Fig. 1 (left) A clique, (center) a 2-plex, and (right) a 3-plex

or neighboring each other. The neighborhood $N(v)$ of v is the set of nodes adjacent to it, and the degree $d(v)$ of v is defined as the number of its neighbors, i.e. $|N(v)|$. An attributed graph is a tuple (G, ϕ, ψ) where $\phi : V \rightarrow \mathbb{R}^{h_V}$ and $\psi : V \times V \rightarrow \mathbb{R}^{h_E}$ are functions that assign a vector of features to each node and to each edge, respectively of size h_V and h_E . If $e \notin E$, then $\psi(e) = \mathbf{0}$. An attributed graph can be also represented in matrix notation (\mathbf{A}, \mathbf{X}) by taking an arbitrary (usually predefined) ordering of its nodes $V = \{v_1, \dots, v_n\}$ and by having $\mathbf{X} \in \mathbb{R}^{n \times h_V}$ as its node-feature matrix, whose rows are defined as $\mathbf{x}_i = \phi(v_i)$. The term $\mathbf{A} \in \mathbb{R}^{n \times n \times h_E}$ denotes its adjacency matrix, where $\mathbf{A}_{ij} = \mathbf{A}_{ji} = \psi(\{v_i, v_j\})$.

A k -plex is a subset of nodes $S \subseteq V$ such that each node in S has at least $|S| - k$ adjacent nodes in S : for all $v \in S$, we have $|N(v) \cap S| \geq |S| - k$. This definition is quite flexible as for $k = 1$ we get the classical clique and for larger values of k we obtain a relaxed and broader family of (possibly larger) subgraphs of G . Some examples are shown in Fig. 1. A k -plex cover of G is a family of subsets \mathcal{S} of V such that each set $S \in \mathcal{S}$ is a k -plex and their union is $\bigcup_{S \in \mathcal{S}} S = V$.

2.2 Preliminaries on graph neural networks

Graph Neural Networks are a specialized kind of neural network that adaptively learns fixed-size representations of a set of graphs in a given distribution. GNNs are typically defined in terms of graph convolutions, both in the spectral (Bruna et al. 2014; Deferrard et al. 2016; Kipf and Welling 2017) and in the spatial domain (Hamilton et al. 2017; Monti et al. 2017; Xu et al. 2018; Veličković et al. 2018; Xu et al. 2019; Morris et al. 2019). Spectral GNNs perform graph convolution using the graph Fourier transform, or by approximating it with a (truncated) Chebyshev expansion (Hammond et al. 2011). Spatial GNNs, instead, perform graph convolution by applying a learned filter to the features of each node and its local neighborhood. Most GNNs can be described as instances of the so-called Message-Passing Neural Network model (Gilmer et al. 2017), which takes inspiration from the classical message-passing paradigm and from the very first learning approaches to graph-structured data (Micheli 2009; Scarselli et al. 2005, 2009; Gori et al. 2005). Battaglia et al. (2018) further abstract this model by proposing a more general form, GRAPHNET, where also edge- and graph-level attributes can be parametrized.

GNNs often leverage on pooling techniques to obtain a coarsened representation of the graphs in input. As in Convolutional Neural Networks (Fukushima 1980; LeCun et al. 1989), pooling serves both as dimensionality reduction and to reduce the distance between objects in the input, thus increasing the receptive field of parametric functions applied on its output. Graph pooling is typically performed using classical clustering

algorithms such as GRACLUS (Dhillon et al. 2007; Bruna et al. 2014; Defferrard et al. 2016; Simonovsky and Komodakis 2017; Ma et al. 2019; Wang et al. 2020), or adaptively, as in Ying et al. (2018); Cangea et al. (2018); Gao and Ji (2019); Lee et al. (2019). A more in-depth comparison will be discussed in Sect. 3. Pooling is also used to obtain a final, global representation of the whole graph in input. This technique, which is usually denoted as global pooling, is usually performed by using standard aggregation functions such as *sum*, *max* or *mean* (Xu et al. 2019), or by exploiting techniques from the field of (multi-)set representation learning, such as GLOBALATTENTIONPOOL (Li et al. 2016), SET2SET (Vinyals et al. 2016), and SORTPOOL (Zhang et al. 2018).

2.3 Graph pooling with k -plexes

KPLEXPOOL computes a k -plex cover $\mathcal{S} = \{S_1, \dots, S_c\}$ of the input graph (G, ϕ, ψ) , for a given k , and returns a coarsened graph (G', ϕ', ψ') , such that

$$V' = V(G') = \{v'_1, \dots, v'_c\}, \quad (2.1)$$

$$E' = E(G') = \{\{v'_i, v'_j\} \mid E(G[S_i, S_j]) \neq \emptyset\}, \quad (2.2)$$

where $E(G[S_i, S_j]) = E(G) \cap (S_i \times S_j)$. Node v'_i represents the coarsened version of S_i , and edge $\{v'_i, v'_j\}$ exists iff there is at least one edge in G linking a node of S_i with a node of S_j . The feature functions $\phi' : V' \rightarrow \mathbb{R}^{h_v}$ and $\psi' : V' \times V' \rightarrow \mathbb{R}^{h_e}$ aggregate, respectively, features belonging to the same k -plex S_i and features of edges linking two different S_i and S_j . In other words, they are defined in such a way to provide a suitable relabeling for nodes and edges in the coarsened graph:

$$\phi'(v'_i) = \beta(\{\phi(v) \mid v \in S_i\}), \quad (2.3)$$

$$\psi'(\{v'_i, v'_j\}) = \gamma(\{\psi(e) \mid e \in E(G[S_i, S_j])\}), \quad (2.4)$$

where β and γ are arbitrary aggregation functions defined over multisets of feature vectors. Typical aggregators for node attributes are *element-wise max* or *sum* (Xu et al. 2019). For edge weights, our choice is the *sum reduction*: if the input graph has $\forall e. \psi(e) = 1$, then the weight of an edge in the coarsened graph is the number of edges crossing the two linked k -plexes. Figure 2 shows an example of a graph and the hierarchical reduction produced by the application of a series of three sum-pooling.

Differently from other partitioning-based graph coarsening methods (Dhillon et al. 2007; Ng et al. 2002), in our approach a node may belong to multiple k -plexes. This is also a key difference between CLIQUEPOOL (Luzhnica et al. 2019) and KPLEXPOOL with $k = 1$ (i.e., performing a *clique cover*), where the former model forces a partition between nodes potentially destroying structural relationships in the communities.

From Eq. 2.2 and by the fact that every edge forms a clique, it follows that whenever two k -plexes share a node, their respective aggregated nodes in the coarsened graph are adjacent. For this reason, not only G' can be denser than G , but KPLEXPOOL may also produce G' such that $e(G') > e(G)$ on some pathological cases (e.g., star graphs). For

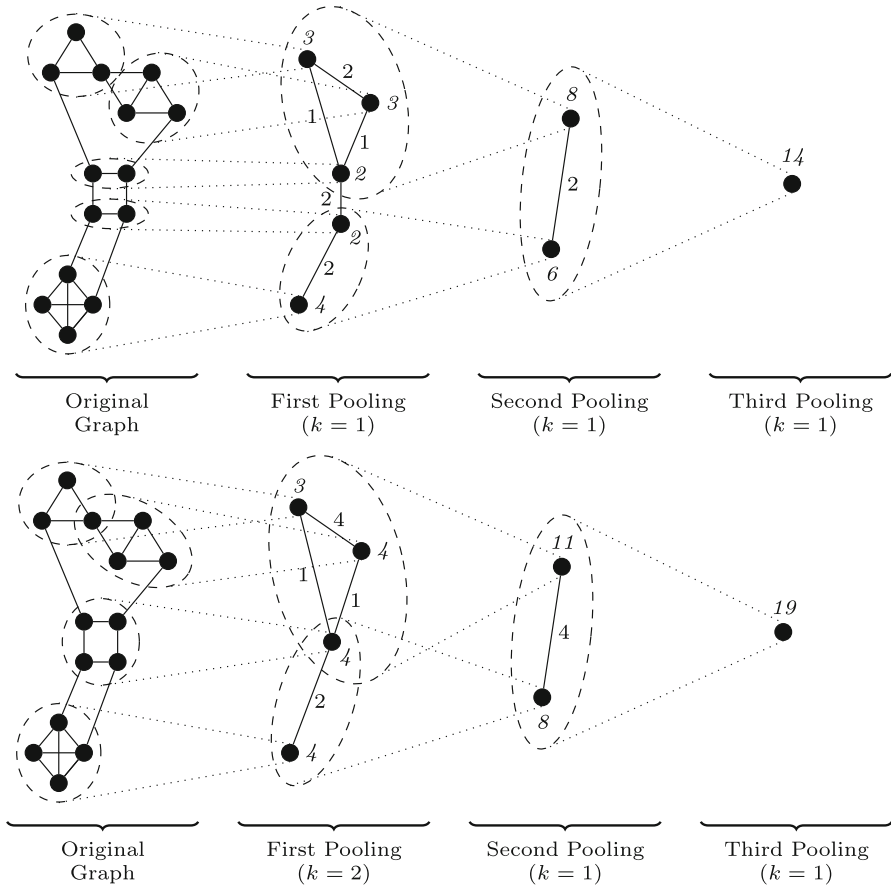


Fig. 2 Two examples sum-pooling on the same graph, where $\phi(v) = 1$ and $\psi(e) = 1$. In the first example (top), the hierarchy is formed by clique pooling (i.e., k -plex pooling with $k = 1$). In the second example (bottom), the first layer of the hierarchy is obtained using 2-plex pooling. Numbers in roman and italic fonts represent edge and node attributes, respectively

this reason, our KPLEXPOOL algorithm incorporates a graph sparsification method, whose details are discussed in Sect. 2.6.

To provide a compact (vectorial) definition of our operator, we can represent the k -plex cover as a matrix $\mathbf{S} \in \{0, 1\}^{n \times c}$ of *hard-assignments*, i.e., $\mathbf{S}_{ij} = \llbracket v_i \in S_j \rrbracket$. If we use *sum* for both aggregation functions, it is easy to see that Eqs. 2.3, 2.4 can be rewritten in matrix form as

$$\mathbf{X}' = \mathbf{S}^T \mathbf{X} \quad \text{and} \quad \mathbf{A}' = \mathbf{S}^T \mathbf{A} \mathbf{S}, \tag{2.5}$$

where $\mathbf{X}' \in \mathbb{R}^{c \times h_V}$ and $\mathbf{A}' \in \mathbb{R}^{c \times c \times h_E}$ are, respectively, the node feature and adjacency matrix of G' .

Algorithm 1 KPLEXCOVER

input A graph G , an integer $k \geq 1$, and two priority functions f and g .
output A k -plex cover \mathcal{S} of G .
1: $\mathcal{S} \leftarrow \emptyset$
2: $U \leftarrow V(G)$
3: **while** $U \neq \emptyset$ **do**
4: $v \leftarrow \operatorname{argmax}_{u \in U} f(u)$
5: $S \leftarrow \text{FINDKPLEX}(G, k, g, v)$
6: $\mathcal{S} \leftarrow \mathcal{S} \cup \{S\}$; $U \leftarrow U \setminus S$
7: Suitably update priority f .

Algorithm 2 FINDKPLEX

input A graph G , an integer $k \geq 1$, a priority function g , and a pivot node v .
output A k -plex S , s.t. $v \in S$.
1: $S \leftarrow \{v\}$; $C \leftarrow N(v)$
2: **while** $C \neq \emptyset$ **do**
3: $u \leftarrow \operatorname{argmax}_{w \in C} g(w)$
4: $S \leftarrow S \cup \{u\}$; $C \leftarrow C \setminus \{u\}$
5: **for** $w \in S$ **do**
6: **if** $|S \setminus N(w)| = k$ **then** $C \leftarrow C \cap N(w)$
7: **for** $w \in C$ **do**
8: **if** $|S \setminus N(w)| = k$ **then** $C \leftarrow C \setminus \{w\}$
9: **for** $w \in N(u)$ **do**
10: **if** $|S \setminus N(w)| < k$ **then** $C \leftarrow C \cup \{w\}$
11: Suitably update priority g .

2.4 K-plex cover algorithm

We propose an algorithm, whose pseudocode is shown in Algorithms 1, 2, that finds a cover containing *large* k -plexes that have *small* intersection. The rationale for this choice is driven by the sought-after effect on graph pooling mechanisms in graph neural networks. On one hand, we seek to condense into a single community-node those neighboring nodes which are likely to share the same context and, hence, very similar embeddings. On the other hand, we would like the pooled graph to preserve diversity for nodes belonging to different communities, i.e. avoiding trivial aggregations which would induce heavy connectivity between the communities. Our algorithm is inspired to the clique covering framework in Conte et al. (2016, 2020), and leverages on heuristics that specifies the order on which nodes are considered for k -plex inclusion. Algorithm 1 receives in input this order by means of two priority functions f , g on V that are defined to provide large k -plexes with small pair-wise intersections. In practice, we fixed f , g to prioritise nodes with lower degree (for f) and more neighbors in the k -plex (for g). A deeper discussion on priority functions is provided in Table 1.

Algorithm 1 uses Algorithm 2 as a subroutine, where the latter, starting from an *uncovered* node v (i.e., a node that is not included in the current cover), retrieves a k -plex $S \subseteq V$ containing v . We discuss more in details both Algorithms 1 and 1 in the following. Algorithm 1 begins by iterating over the available nodes in the candidate set U , which is initialized with the whole set of nodes of the input graph. At each iteration, it selects the next candidate v by extracting the node in U with highest

Table 1 Proposed priority functions for f and g , where π_n represents a random permutation of the first n natural numbers

	Priority	Value	Update cost
f	Random	$\pi_n(v)$	–
	Max-degree	$d(v)$	–
	Max-uncovered	$ N(v) \setminus \bigcup_{S \in \mathcal{S}} S $	$O(d(v))$
g	Max-in-k-plex	$ N(v) \cap S $	$O(d(v))$
	Max-candidates	$ N(v) \cap C $	$O(d(v))$

Note that every f is also a valid g , but not the other way round. The third column shows the computational cost of updating a priority for every iteration of the while loop in Algorithm 2

priority $f(v)$. The node v will be then used as a starting node for retrieving the next k -plex $S \subseteq V$, using Algorithm 2. The elements of S will then be removed from the set U of candidates, and S will be included in the output cover \mathcal{S} . Note that the nodes in the k -plexes are removed from U but not from the graph, hence successive executions of Algorithm 2 may contain previously removed nodes. The algorithm stops when eventually all the nodes are removed from U , and then returns the cover \mathcal{S} . Algorithm 2, instead, constructs a k -plex S starting from a given node v , which is the only element available at startup. It initializes a candidate set C of nodes that could be part of the k -plex, this time relying on $N(v)$. Again, Algorithm 2 iterates over the nodes in C following the ordering defined by the priority g , and adds them to S . The main loop has the following invariants:

$$\forall u \in S : |S \setminus N(u)| \leq k , \tag{2.6}$$

$$\forall u \in S : |S \setminus N(u)| = k \implies C \setminus N(u) = \emptyset , \tag{2.7}$$

where Eq. 2.6 states that every node u in S needs to have at least $|S| - k$ adjacent nodes in S and Eq. 2.7 states that, when u has exactly $|S| - k$ adjacent nodes in S , we cannot have nodes in C that are *not* adjacent to u (as it would break Eq. 2.6 if selected). As a result, *any* node from C can be added to S .

Both invariants are satisfied at the first iteration as all the candidates in $C = N(v)$ are adjacent to v , which is the only element in S . At each iteration, Algorithm 2 preserves Eq. 2.6 by selecting a node u from C according to priority g . It needs to preserve Eq. 2.7 as S changes because of the addition of u . The first two for loops remove the nodes from C that no longer can be added to S . The third loop adds to C the nodes adjacent to u which can be added to S . After that, g is updated.

The above invariants guarantee that any S retrieved by Algorithm 2 is a k -plex. As Algorithm 1 terminates only when all the nodes have been covered by at least one k -plex in the current solution \mathcal{S} , we obtain that \mathcal{S} is a k -plex cover, hence proving the correctness of Algorithm 1.

2.5 Priority functions

To effectively summarize the graph for pooling purposes, we need to cover all the nodes with as few k -plexes as possible: we thus want these k -plexes to be large and with small overlaps. Following the experimented heuristics in Conte et al. (2016, 2020), we adapt the best performing priorities to the case of k -plexes. Specifically, the priorities f and g define how the nodes will be extracted during the execution of Algorithms 1, 2, guiding the covering process.

Table 1 provides a selection of priority functions along with their computational cost. They span from random baseline policies to orders induced by the topology of the pooled graph and intended to yield pooled graph with interesting structural properties. *Random* and *max-degree* priorities should be quite self-explanatory; *max-uncovered* returns the number of neighbors that are not yet covered by a k -plex in \mathcal{S} . Every f priority is also a valid strategy for g . Concerning the latter, we can also consider

- i *max-in- k -plex*, that assigns to every candidate node the number of its neighbors within the current k -plex S ; and
- ii *max-candidates*, that assigns to every candidate node the number of its neighbors within the current candidate set C .

For the experimental assessment in Sect. 4, we implement the cover priority f as the concatenation of *min-degree* and *max-uncovered*. Here the term concatenation denotes the fact that nodes are ordered following a lexicographic ordering defined first on *min-degree* with ties broken on *max-uncovered* (note that *min-degree* is discrete valued and many nodes are likely to have the same priority). The k -plex priority g is obtained as the concatenation of *max-in- k -plex*, *max-candidates* and *min-uncovered*, following the lexicographic ordering approach described for f . Both of the concatenated priorities, respectively for f and g , will be referred to as *default* in the following. Note that *min-* priorities can be trivially obtained by the respective *max-* definition. The choice of these combinations of policies is driven by the observation that they are able to generate large k -plexes while maintaining small intersections between them which, again, is a key property we would like to induce in our pooling mechanism. Figures 3, 4 show two useful statistics obtained executing Algorithm 1 with various combinations of the proposed priority functions. Specifically:

- the *average number of clusters* in a cover (Fig. 3), which we aim to minimize, since a high number of clusters can be a signal of large intersections and/or small k -plexes;
- the *average number of occurrences of nodes* in a cover (Fig. 4), which again we want to minimize, since a node has more than one occurrence if it belongs to multiple k -plexes and, thus, lies in an intersection between them.

2.6 Cover post-processing

On certain kinds of graphs, like star graphs or scale-free networks, few *hub* nodes have a degree that greatly exceeds the average on the graph: Algorithm 1 may include them in many distinct (and all pairwise adjacent) k -plexes, generating dense artifacts

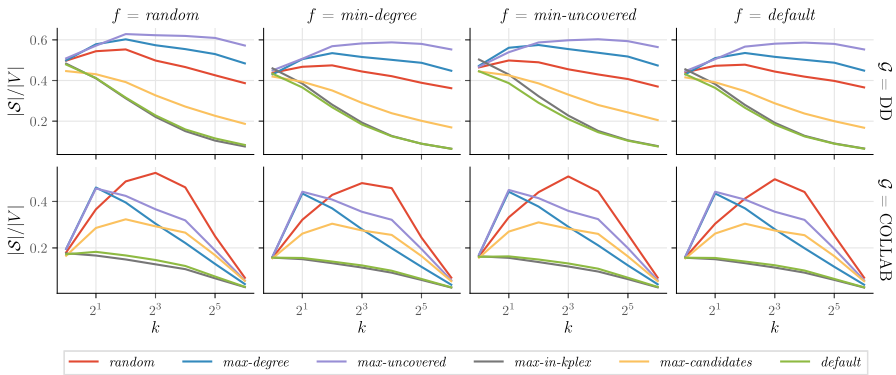


Fig. 3 Average number of k -plexes obtained executing Algorithm 1 on 100 graphs of DD and COLLAB using different combination of k , f (one per column) and g (depicted with different line colors). The number of clusters $|S|$ is expressed with respect to the number of nodes $|V|$ in the original graph (Color figure online)

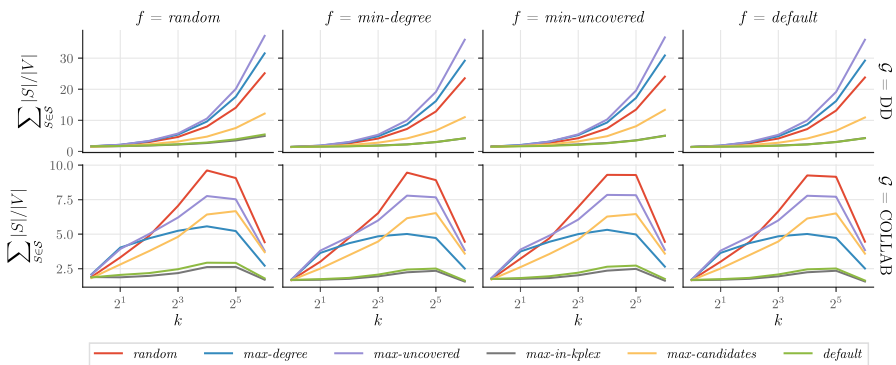


Fig. 4 Average number of occurrence of a node in the k -plexes of a cover obtained executing Algorithm 1 on 100 graphs of DD and COLLAB using different combination of k , f (one per column) and g (depicted with different line colors). The number of occurrences are expressed as $\sum_{S \in \mathcal{S}} |S|/|V| \geq 1$ (Color figure online)

that do not well represent the graph. To overcome this problem, we discuss a cover post-processing mechanism, referred to as *hub promotion*, that sparsifies the coarsened graph by changing the cover assignments, removing hub nodes from the sets of the cover and assigning each to a dedicated singleton set. Specifically, for a given cover \mathcal{S} of a graph G , and a threshold value $p \in [0, 100]$, we proceed as follows.

1. Compute the covering index $s(v) = |\{S \in \mathcal{S} \mid v \in S\}|$ for every $v \in V$.
2. Compute s_p as the p -th percentile of the above values and find the subset of hub nodes $H_p \subseteq V$ such that $s(v) > s_p$ for all $v \in H_p$.
3. Generate a modified cover $\mathcal{S}_p = \{S \setminus H_p \mid S \in \mathcal{S}\} \cup \{\{v\} \mid v \in H_p\}$. Note that, as a limit case, $\mathcal{S}_{100} = \mathcal{S}$.
4. Coarsen G applying Eqs. 2.1, 2.2, 2.3, 2.4 with the resulting cover \mathcal{S}_p .

The effect of this method is assessed on social network datasets in Sect. 4. The results of a detailed ablation study, including both molecular and social graphs, is reported in Sect. 4.1.

2.7 Computational cost

It can be easily shown that Algorithm 2 takes $O(m)$ time, since we can efficiently store and update g with standard data structures: Indeed, as each node v is added to S or C and removed from C at most once, and each update costs $O(d(v))$, the amortized cost is $O(\sum_{v \in V(G)} d(v)) = O(m)$ time. The time cost of Algorithm 1 is bounded by $O(mn)$: each time FINDKPLEX is called at least one new node (v) is covered, thus the **while** loop will be executed $O(n)$ times. Each time, updating S , U , and f , and calling Algorithm 2, all $O(m)$ time, meaning Algorithm 1 always terminates within $O(mn)$ time. The cost of the post-processing is also dominated by the cost of Algorithm 1.

KPLEXPOOL performs ℓ pooling steps. In each step, it finds a cover via Algorithm 1, taking $O(nm)$ time, then performs feature aggregating in the clusters by multiplying two matrices of sizes at most $n \times n$ (the cover has at most n k -plexes). The best known cost for such multiplication is $O(n^\omega)$, where $\omega < 2.373$ (Le Gall 2014), giving a total cost of $O((nm + n^\omega)\ell)$ time.

3 Related works

Models in literature can be partitioned in two general classes, based on whether they tackle the structure reduction problem using a *topological* or an *adaptive* approach (Bacciu et al. 2020). The former exploits solely structural and community information conveyed by the sample topology. The latter generates the graph reduction using the information on the node embeddings, leveraging adaptive parameters that are trained together with graph convolution parameters. Several topological pooling algorithms are based on a clustering of the adjacency matrix. METIS (Karypis and Kumar 1998) and GRACLUS (Dhillon et al. 2007) are multi-level clustering algorithms that partition the graph by a sequence of coarsening and refinement phases. LOUVAIN (Blondel et al. 2008), ECG (Poulin and Th  berge 2019), and LEIDEN (Traag et al. 2019), start with singleton clusters and then greedily merge the ones that maximize the overall modularity. NMFPOOL (Bacciu and Di Sotto 2019) performs a probabilistic clustering of the nodes by non-negative factorization of the adjacency matrix. Other approaches leverage node neighborhoods and local communities, such as Gama et al. (2019). ECC (Simonovsky and Komodakis 2017) considers a point cloud representation of the graph to group nearby nodes in Euclidean space. EIGENPOOL (Ma et al. 2019) and HAARPOOL (Wang et al. 2020; Li et al. 2020) define novel aggregation techniques on the top of existing clustering algorithms (such as METIS). CLIQUEPOOL (Luzhnica et al. 2019), instead, aggregates the attribute vectors of the nodes within the same clique. Adaptive pooling is, again, largely based on clustering but, differently from topological approaches, this is performed on the neural embeddings obtained from the graph convolutional layers. These methods rely on a parameterized

clustering of the neural activations which entails that they need to preserve differentiability. DIFFPOOL (Ying et al. 2018) has pioneered the approach by introducing a hierarchical clustering that soft-assigns each node to a fixed number of clusters c , generating $\mathbf{S} \in \mathbb{R}^{n \times c}$ with another GNN. This process does not yield to an actual reduction of the adjacency matrix, as this would infringe differentiability, resulting in highly parameterized models of considerable complexity. MINCUTPOOL (Bianchi et al. 2020) further improve this technique by optimizing a relaxed version of the normalized-cut objective. STRUCTPOOL (Yuan and Ji 2019), instead, computes the soft-clustering associations through a Conditional Random Field (CRF). GPOOL (Gao and Ji 2019; Cangea et al. 2018), also known as TOPKPOOL (Fey and Lenssen 2019), addresses this shortcoming by learning a single vector $\mathbf{p} \in \mathbb{R}^{h_{in}}$, used to compute *projection scores* that serve to retain the top $\lceil kn \rceil$ ranked nodes. SAGPOOL (Lee et al. 2019; Knyazev et al. 2019) later extended TOPKPOOL to compute the score vector by means of a GNN. GSAPOOL (Zhang et al. 2020) further extends this model by a convex combination of two projection vectors: one learned by a GNN, and another by a classical neural network (with no topology information). The authors of ASAPool (Ranjan et al. 2020) showed the limitations of using a standard GCN (Kipf and Welling 2017) to compute the projection scores, and defined another convolution for graphs (LECONV) for that specific purpose. EDGEPOOL (Diehl 2019; Diehl et al. 2019) reduces the graph by *edge contraction*, based on a score computed by a neural model that takes the edge incident nodes as input.

KPLEXPOOL falls into the family of topological pooling but proposes a radically different approach to adjacency clustering models, which is based on well-grounded concepts from graph algorithmics. CLIQUEPOOL is the most closely related model, but it considers a much more restricted and less flexible form of community than ours. Also, CLIQUEPOOL is limited to simple graph partitions, whereas our approach can leverage the flexibility of assignments of graph covers. The effect of the greater generality and flexibility of KPLEXPOOL is evident by its excellent empirical performances on a variety of graph topologies (Sect. 4). When compared to adaptive pooling methods, KPLEXPOOL has certainly the disadvantage of relying on graph reductions that are not driven by the predictive task. Nonetheless, the experimental assessment shows that KPLEXPOOL can achieve state of the art performances on both molecular and social graphs, also outperforming adaptive approaches where pooling is learned to optimize the predictive task.

4 Experimental analysis

We tested KPLEXPOOL against related methods from literature on four molecular graph datasets, namely DD (Dobson and Doig 2003), NCI-1 (Wale et al. 2008), ENZYMES (Schomburg et al. 2004), and PROTEINS (Borgwardt et al. 2005), and five social network datasets, COLLAB, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, and REDDIT-MULTI-5K (Yanardag and Vishwanathan 2015). All datasets have been retrieved from the TU-Dortmund collection (Morris et al. 2020).

For fairness, each pooling method has been tested by plugging it into the same standardized architecture (BASELINE), comprising ℓ convolutional blocks, followed

Table 2 Hyper-parameter (HP) space used in the grid-search

Model	HP	Values
All	GNN	GCN, GRAPHSAGE
	ℓ	2, 3
	η	1, 0.5, 0.2, $0.1 (\times 10^{-3})$
	h	64, 128
KPLEXPOOL	k	1, 2, 4, 8
TOP-, SAG-, DIFF-, MINCUTPOOL	r	1/4, 1/2, 3/4

GNN is the convolution type, ℓ the number of layers, η the learning rate, h the hidden size, r the reduction factor, and k the k -plex value

by two dense layers, the latter interleaved by dropout with probability 0.3. Every convolutional block is formed by two GNN layers (either GCN (Kipf and Welling 2017) or GRAPHSAGE (Hamilton et al. 2017)) with Jumping Knowledge (Xu et al. 2018) followed by a dense layer. After every convolutional block we have a global sum-pooling, and the concatenation of their resulting vector is batch-normalized and fed to the final dense block. Every layer, GNN or dense, has h units and a ReLU activation function (Goodfellow et al. 2016). For every other model, a pooling layer is placed after the first $\ell - 1$ convolutional blocks and its output feed to the next block. All models have been implemented using PyTorch-Geometric (Fey and Lenssen 2019), which also provided implementations for GRACLUS, DIFFPOOL, MINCUTPOOL, TOPKPOOL, and SAGPOOL. LEIDEN has instead been computed using the cuGraph library (RAPIDS Development 2018). We re-implemented CLIQUEPOOL using the NetworkX library (Hagberg et al. 2008), which provided an implementation of the Bron and Kerbosch (1973) algorithm and its optimizations (Tomita et al. 2006; Cazals and Karande 2008).

Our experimental approach followed the standardized reproducible setting in Errica et al. (2020). Specifically, for model assessment, we used a stratified 10-fold cross-validation and for model selection an inner stratified shuffle split, generating a validation set of the same size of the outer fold and leaving the remaining examples as training set. We performed a grid search for each fold, using the parameter space listed in Table 2. For each parameter combination, we trained each model with early-stopping after 20 epochs, monitored on the validation set. We selected the one that obtained the highest accuracy on the validation set evaluated it on the outer fold. We tested different configurations of KPLEXPOOL depending on the domain. Since graphs become denser after each pooling layer, on biological datasets we tested the effect of a progressive reduction factor $r_k \in (0, 1]$, so that pooling at layer ℓ has $k^{(\ell)} = \lceil r_k k^{(\ell-1)} \rceil$. Here we also used the concatenation of *sum* and *max* as aggregation function on non-parametric pooling methods, i.e. LEIDEN, GRACLUS, CLIQUEPOOL and KPLEXPOOL. This could not be applied to selection-based methods (TOPKPOOL, SAGPOOL), nor to the soft-clustering in DIFFPOOL and MINCUTPOOL.

For the sake of conciseness, in the following, we summarize the empirical results that assess the effect of the post-processing technique described in Sect. 2.6. Section

Table 3 Test classification accuracy on chemical benchmarks (mean \pm std)

	DD	ENZYMES	NCI-1	PROTEINS
BASELINE	74.79 \pm 3.08	43.00 \pm 10.82	78.08 \pm 2.38	71.61 \pm 5.35
LEIDEN	74.96 \pm 2.70	46.00 \pm 5.97	77.69 \pm 2.11	74.04 \pm 4.29
GRACLUS	77.42 \pm 3.45	42.67 \pm 7.82	78.06 \pm 2.39	74.12 \pm 3.36
CLIQUEPOOL	74.88 \pm 4.35	42.17 \pm 7.07	78.83 \pm 1.82	73.86 \pm 3.58
TOPKPOOL	73.35 \pm 4.29	39.17 \pm 9.79	74.09 \pm 7.29	74.12 \pm 4.05
SAGPOOL	74.75 \pm 3.10	37.67 \pm 10.23	78.01 \pm 1.68	73.31 \pm 4.54
DIFFPOOL	OOB	46.00 \pm 9.17	76.76 \pm 2.37	75.02 \pm 4.14
MINCUTPOOL	OOB	40.67 \pm 8.67	74.82 \pm 5.12	75.12 \pm 2.97
KPLEXPOOL	77.76 \pm 2.92	39.67 \pm 7.52	79.17 \pm 1.73	75.11 \pm 2.80
– with $r_k = 0.5$	75.98 \pm 3.28	43.33 \pm 6.58	78.08 \pm 1.97	75.92 \pm 3.88

Bold highlights the best performing model. OOB stands for *out of resources*, and r_k is the incremental reduction factor for k

4.1 further provides an ablation study showing how k , the progressive reduction factor r_k , and the hub promotion threshold contribute to the result.

All models were trained on a NVIDIA[®] V100 GPU with 16GB of dedicated memory, while the coverings were pre-computed on CPU, a Intel[®] Xeon[®] Gold 6140M with 1.23TB of RAM. KPLEXPOOL has been implemented both in sparse coordinate form, which is slower but space efficient, and in matrix form, which is space-inefficient but apt to GPU parallelization. In the experiments, we used the latter except for DD, REDDIT-B and REDDIT-5K, as they contain larger graphs. For the same reason, DIFFPOOL could not be trained on these datasets, since it required too much memory unless using only six samples per batch, thus requiring longer training times (more than two weeks per experiment). For the experiments in which DIFFPOOL hits the out-of-resource limit, we report results from Errica et al. (2020), which have been obtained under similar, although not fully equivalent, conditions. The coarsened graphs' topologies for non-parametric methods were pre-computed once at the beginning of every experiment, while their node features were aggregated at every training step.

Tables 3, 4 show the mean accuracy and standard deviation on test data (outer fold). On chemical datasets, KPLEXPOOL yields competitive results on all datasets, with higher performances than other related methods on all benchmarks but ENZYMES. The application of the incremental reduction r_k to the k value provides sensible benefits only on ENZYMES, while on other tasks effects are superficial at most.

On social benchmarks, KPLEXPOOL performs better than parametric pooling models on all datasets, when considering the same experimental conditions. DIFFPOOL is out-of-resources on REDDIT data, but KPLEXPOOL is still competitive also with respect to DIFFPOOL results from Errica et al. (2020). On REDDIT-5K, only the topological pooling of GRACLUS yields to higher accuracy. Applying *hub-promotion* (Sect. 2.6) increases KPLEXPOOL performance on IMDB-M and REDDIT-B, as shown in Table 4. Its community-seeking bias seems certainly very adequate for the processing of social graphs, where adaptive pooling methods do not seem capable of leveraging their parameters to produce more informative graph reductions. Perhaps surprisingly,

Table 4 Test classification accuracy on social network benchmarks (mean \pm std)

	COLLAB	IMDB-B	IMDB-M	REDDIT-B	REDDIT-5K
BASELINE	74.44 \pm 2.55	69.20 \pm 6.92	47.20 \pm 4.27	84.85 \pm 6.70	51.71 \pm 3.12
LEIDEN	72.88 \pm 1.88	69.30 \pm 4.08	47.53 \pm 4.45	81.05 \pm 4.46	51.73 \pm 3.31
GRACLUS	72.80 \pm 1.10	68.70 \pm 4.45	47.20 \pm 2.93	86.85 \pm 2.84	53.77 \pm 1.29
CLIQUEPOOL	75.90 \pm 2.08	70.00 \pm 3.71	47.87 \pm 3.34	85.45 \pm 3.52	54.43 \pm 2.19
TOPKPOOL	73.30 \pm 2.96	68.20 \pm 7.81	46.93 \pm 3.03	78.60 \pm 4.30	50.33 \pm 2.84
SAGPOOL	73.40 \pm 2.47	65.40 \pm 6.28	46.33 \pm 3.68	80.15 \pm 7.49	49.79 \pm 2.63
DIFFPOOL	70.92 \pm 2.95	68.80 \pm 8.04	47.07 \pm 1.73	OOOR	OOOR
MINCUTPOOL	73.16 \pm 1.95	70.60 \pm 3.93	44.60 \pm 4.08	OOOR	OOOR
KPLEXPOOL	76.20 \pm 2.08	72.00 \pm 4.78	46.60 \pm 4.08	86.45 \pm 3.50	50.65 \pm 3.41
– with $p = 95$	75.98 \pm 2.31	69.40 \pm 4.88	48.73 \pm 4.33	87.90 \pm 3.67	51.37 \pm 2.77

Bold highlights the best performing model. OOR stands for *out of resources*, and p is the percentile threshold value used for *hub-promotion*

Table 5 Average rank of every model (the lower the better) computed using both KPLEXPOOL's *plain* (i.e., with $r_k = 1$ and $p = 100$), *best* and *worst* results on *chemical*, *social*, and *all* datasets for which all results were available (i.e., all except DD, REDDIT-B, and REDDIT-5K)

	Plain			Best			Worst		
	Ch.	Soc.	All	Ch.	Soc.	All	Ch.	Soc.	All
BASELINE	5.00	3.67	4.33	5.33	4.00	4.67	5.00	3.67	4.33
LEIDEN	4.33	4.33	4.33	4.33	4.67	4.50	4.33	4.33	4.33
GRACLUS	4.00	6.00	5.00	4.33	6.33	5.33	4.00	6.00	5.00
CLIQUEPOOL	4.67	2.00	3.33	5.00	2.33	3.67	4.33	1.67	3.00
TOPKPOOL	7.00	6.33	6.67	7.00	6.67	6.83	7.00	6.33	6.67
SAGPOOL	7.33	7.00	7.17	7.33	7.00	7.17	7.33	7.00	7.17
DIFFPOOL	3.67	6.67	5.17	3.67	7.00	5.33	3.67	6.67	5.17
MINCUTPOOL	5.00	5.67	5.33	5.67	5.67	5.67	5.00	5.33	5.17
KPLEXPOOL	3.33	3.00	3.17	1.67	1.00	1.33	3.67	3.67	3.67

Bold highlights the best ranking models

KPLEXPOOL achieves excellent performances also on molecular data, where we would have expected adaptive models to have an edge, confirming our initial intuition about the flexibility and generality of k -plex cover communities. These results can be well appreciated in Table 5, where we report the average rank of each model separately on chemical, social, and the union of all datasets, with respect to the results shown in Tables 3, 4. For KPLEXPOOL, we report the average rank considering a *plain* configuration (i.e. no incremental reduction nor hub-promotion) as well as for the best and worst performing configurations. These results highlight how the performances of KPLEXPOOL are remarkably stable with respect to the choice of its configuration options (cover post-processing methods).

Table 6 Test classification accuracy (expressed as mean \pm std) obtained on ENZYMES, NCI-1, and PROTEINS with different combinations of k and r_k , using KPLEXPOOL with $\ell = 3$, $h = 64$, and GNN = GCN fixed

Dataset	k	$r_k = 1$	$r_k = 1/2$	$r_k = 1/4$	$r_k = 1/8$
ENZYMES	1	37.83 \pm 6.58	–	–	–
	2	39.83 \pm 7.13	42.17 \pm 8.98	–	–
	4	37.83 \pm 6.83	34.33 \pm 7.75	35.33 \pm 5.72	–
	8	36.67 \pm 9.94	38.00 \pm 7.45	37.67 \pm 5.44	36.67 \pm 8.47
NCI-1	1	77.88 \pm 1.58	–	–	–
	2	76.98 \pm 1.78	77.11 \pm 1.61	–	–
	4	76.28 \pm 2.25	77.23 \pm 2.50	77.30 \pm 2.11	–
	8	76.69 \pm 2.11	76.48 \pm 2.72	77.25 \pm 2.49	77.59 \pm 2.71
PROTEINS	1	74.93 \pm 3.56	–	–	–
	2	75.38 \pm 3.22	74.84 \pm 4.13	–	–
	4	75.11 \pm 4.28	75.29 \pm 3.48	74.30 \pm 3.78	–
	8	72.87 \pm 4.67	74.39 \pm 4.27	73.58 \pm 3.26	74.20 \pm 5.31

A “–” implies the experiment is equivalent to the closest on its left, as the smaller r_k value reduced k in the same manner

4.1 Ablation studies and practical considerations

For completeness, we performed ablation studies aimed at analyzing the every component of KPLEXPOOL contributes to the overall performance. Table 6 compares the results obtained on ENZYMES, NCI-1, and PROTEINS using KPLEXPOOL with different combinations of k and r_k . We used the experimental approach described in Sect. 4, but restricted the hyper-parameter space (Table 2) by fixing $\ell = 3$, $h = 64$, and GNN = CGN. As anticipated in Sect. 4, KPLEXPOOL appears to yield a better accuracy by aggregating 2- to 4-plexes instead of simple cliques, and the best values (highlighted in bold) are obtained for $k = 2$ on two out of the three datasets.

Applying a reduction factor seem to be more effective with larger k values: This could be caused by the fact that a large k generates a cover containing few, large, sets; hence, pooling layers beyond the first one will work on a smaller and denser coarsened graph, with denser inner communities, that is better summarized by cliques or k -plexes with small k values.

Table 7 compares instead the results obtained on COLLAB, IMDB-B, and IMDB-M using the same approach as above, but varying k and p (Sect. 2.6), and fixing $\ell = 2$.

Considering the generation process of the networks (see Yanardag and Vishwanathan 2015), we note that COLLAB is obtained by connecting authors that co-authored a paper, while IMDB-B and IMDB-M by connecting actors/actresses that co-starred in a movie. Furthermore, IMDB-M includes more movies than IMDB-B. We can thus imagine that IMDB-M undergoes a more prominent presence of hubs compared to IMDB-B due to the “rich gets richer” phenomenon (i.e., if we increase the number of movies considered, more famous actors have a greater probability of being featured in these movies and thus receiving more connection).

Table 7 Test classification accuracy (expressed as mean \pm std) obtained on COLLAB, IMDB-B and IMDB-M with different combinations of k and p , using KPLEXPOOL with $\ell = 2$, $h = 64$, and GNN = GCN fixed

Dataset	k	$p = 100$	$p = 95$	$p = 90$	$p = 85$	$p = 80$
COLLAB	1	76.16 \pm 1.96	76.28 \pm 1.10	74.82 \pm 2.17	74.80 \pm 1.85	75.24 \pm 1.47
	2	75.70 \pm 1.97	75.58 \pm 1.87	75.22 \pm 2.51	74.08 \pm 2.23	75.22 \pm 2.28
	4	75.32 \pm 1.84	75.30 \pm 1.89	75.50 \pm 1.62	75.32 \pm 2.29	74.68 \pm 1.98
	8	74.32 \pm 1.84	73.50 \pm 1.71	73.54 \pm 1.96	74.42 \pm 1.71	74.32 \pm 2.06
IMDB-B	1	67.50 \pm 7.00	71.20 \pm 3.94	72.60 \pm 4.00	70.80 \pm 4.87	71.90 \pm 3.75
	2	68.60 \pm 6.36	71.50 \pm 4.01	70.80 \pm 3.54	69.40 \pm 7.13	70.70 \pm 3.07
	4	70.70 \pm 4.24	70.70 \pm 3.77	66.90 \pm 8.98	70.70 \pm 5.04	68.60 \pm 7.03
	8	68.50 \pm 5.10	70.10 \pm 5.07	67.70 \pm 7.16	64.40 \pm 8.90	68.30 \pm 5.50
IMDB-M	1	48.07 \pm 3.81	47.60 \pm 4.01	46.80 \pm 3.46	47.80 \pm 2.35	47.93 \pm 1.78
	2	48.40 \pm 3.49	46.93 \pm 4.01	47.20 \pm 3.21	47.53 \pm 2.92	48.00 \pm 3.11
	4	46.73 \pm 3.73	47.67 \pm 3.77	47.93 \pm 2.64	47.93 \pm 2.74	48.27 \pm 4.19
	8	47.07 \pm 2.72	47.60 \pm 2.43	46.53 \pm 2.32	48.07 \pm 2.37	47.47 \pm 3.28

This is reflected on the data in Table 7: we can observe how the best value for IMDB-M is obtained when $p = 80$ (i.e., the top 20% of nodes is considered a hub) whereas the best value of IMDB-B is obtained for $p = 90$ (i.e., the top 10% is considered a hub). More in general, we can observe that Table 7 motivates the usage of the hub promotion parameter p , as in all three datasets considered we obtain benefits by setting it as a non-trivial value (we recall that $p = 100$ corresponds to not using hub promotion).

As for varying the k values, we do not observe any significant trend. This is perhaps unsurprising, considering that these datasets are obtained by turning the co-authors of a paper (resp., co-stars of a movie) into a clique, thus $k = 1$ may be general enough for our requirements in some cases, although we observe that the best results are found on different lines for different values of p , and in particular the best value of IMDB-M is found for $k = 4$.

For an optimal choice of parameters, the ideal strategy would be to find the most effective values of k from the hyper-parameter tuning process. However, we observe that the choice $k = 2$ seems to have good overall performance, being often the best result or close to it: if it is required to use the approach *on the fly*, as a rule of thumb we suggest trying low values of k .

5 Conclusions

We have introduced KPLEXPOOL, a novel graph pooling methodology leveraging k -plexes and graph covers. Starting from consolidated graph-theoretic concepts and recent results on scalable community detection (Conte et al. 2018), we have built a flexible graph reduction approach that works effectively across structures with different topological properties. We have provided a general formulation that can account for different node inspection schedules and that can, in principle, be tailored based on

prior or domain knowledge. Nevertheless, in the paper, we discussed the effectiveness of the approach for a fixed node ordering heuristic and cover post-processing strategy, showing the effectiveness of the method even in the plainest configuration.

The resulting KPLEXPOOL algorithm has state-of-the-art performances in 7 out of 9 graph classification benchmarks. KPLEXPOOL is shown to be the best performing method, on average, when confronted with related pooling mechanisms from literature. It does so through a fully topological approach that does not leverage task information for community building. None of the related models, including the adaptive ones, seem to have the same ability to cope effectively with structures of radically different nature (molecules and social networks). Apart from predictive performance, KPLEXPOOL has a very practical advantage in terms of computational cost, when compared to adaptive models such as DIFFPOOL. For instance, its graph reduction can be pre-computed once for the whole dataset and re-used throughout the whole model selection and validation phase, as it does not depend on adaptive node embedding. This aspect is clear from the empirical results, in which DIFFPOOL is shown to fail to complete training within the 2 weeks limit (or to exceed the available GPU memory) on datasets comprising larger graphs. Conversely, as the proposed k -plex cover algorithm is mostly sequential, computing KPLEXPOOL on the fly during the training loop will produce an overhead due to GPU-CPU synchronizations. To overcome this limitation, we need to design a parallel alternative for our algorithm that could also run on GPU. We left this as a future work.

Acknowledgements This work partially supported by the EU H2020 project TAILOR (n.952215), and MIUR under PRIN Project AHeAD (n.20174LF3T8).

Funding Open access funding provided by Università di Pisa within the CRUI-CARE Agreement.

Code availability Code available at <https://github.com/flandolfi/kplex-pool/>.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Bacciu D, Di Sotto L (2019) A non-negative factorization approach to node pooling in graph convolutional neural networks. In: Alviano M, Greco G, Scarcello F (eds) AI*IA 2019-advances in artificial intelligence, Lecture notes in computer science. Springer, Cham, pp 294–306, https://doi.org/10.1007/978-3-030-35166-3_21
- Bacciu D, Errica F, Micheli A (2018) Contextual graph markov model: a deep and generative approach to graph processing. In: International Conference on Machine Learning, pp 294–303, ISSN: 1938-7228
- Bacciu D, Errica F, Micheli A, Podda M (2020) A gentle introduction to deep learning for graphs. Neural Netw 129:203–221. <https://doi.org/10.1016/j.neunet.2020.06.006>
- Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, Raposo D, Santoro A, Faulkner R, Gulcehre C, Song F, Ballard A, Gilmer J, Dahl G, Vaswani A, Allen K,

- Nash C, Langston V, Dyer C, Heess N, Wierstra D, Kohli P, Botvinick M, Vinyals O, Li Y, Pascanu R (2018) Relational inductive biases, deep learning, and graph networks. [arXiv:1806.01261](https://arxiv.org/abs/1806.01261)
- Bianchi FM, Grattarola D, Alippi C (2020) Spectral clustering with graph neural networks for graph pooling. *Proc Int Conf Mach Learn* 1
- Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. *J Stat Mech Theory Exp*. <https://doi.org/10.1088/1742-5468/2008/10/P10008>
- Borgwardt KM, Ong CS, Schönauer S, Vishwanathan SVN, Smola AJ, Kriegel HP (2005) Protein function prediction via graph kernels. *Bioinform (Oxford, Engl)* 21(Suppl 1):i47–56. <https://doi.org/10.1093/bioinformatics/bti1007>
- Bron C, Kerbosch J (1973) Algorithm 457: finding all cliques of an undirected graph. *Commun ACM* 16(9):575–577. <https://doi.org/10.1145/362342.362367>
- Bruna J, Zaremba W, Szlam A, LeCun Y (2014) Spectral networks and locally connected networks on graphs. In: Bengio Y, LeCun Y (eds) *Proceedings of the 2nd international conference on learning representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings*
- Cangea C, Veličković P, Jovanović N, Kipf T, Liò P (2018) Towards sparse hierarchical graph classifiers. [arXiv:1811.01287](https://arxiv.org/abs/1811.01287)
- Cazals F, Karande C (2008) A note on the problem of reporting maximal cliques. *Theor Comput Sci* 407(1):564–568. <https://doi.org/10.1016/j.tcs.2008.05.010>
- Conte A, Grossi R, Marino A (2016) Clique covering of large real-world networks. In: *Proceedings of the 31st annual ACM symposium on applied computing, association for computing machinery, Pisa, Italy, SAC '16*, pp 1134–1139. <https://doi.org/10.1145/2851613.2851816>
- Conte A, De Matteis T, De Sensi D, Grossi R, Marino A, Versari L (2018) D2K: scalable community detection in massive networks via small-diameter k-plexes. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining, association for computing machinery, London, United Kingdom, KDD '18*, pp 1272–1281. <https://doi.org/10.1145/3219819.3220093>
- Conte A, Grossi R, Marino A (2020) Large-scale clique cover of real-world networks. *Inform Comput* 270: <https://doi.org/10.1016/j.ic.2019.104464>
- Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: *Proceedings of the 30th international conference on neural information processing systems, Curran Associates Inc., Red Hook, NY, USA, NIPS'16*, pp 3844–3852
- Dhillon IS, Guan Y, Kulis B (2007) Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Trans Pattern Anal Mach Intel* 29(11):1944–1957. <https://doi.org/10.1109/TPAMI.2007.1115>
- Diehl F (2019) Edge contraction pooling for graph neural networks. [arXiv:1905.10990](https://arxiv.org/abs/1905.10990)
- Diehl F, Brunner T, Le MT, Knoll A (2019) Towards graph pooling by edge contraction. In: *ICML 2019 workshop on learning and reasoning with graph-structured data*
- Dobson PD, Doig AJ (2003) Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol* 330(4):771–783. [https://doi.org/10.1016/s0022-2836\(03\)00628-4](https://doi.org/10.1016/s0022-2836(03)00628-4)
- Errica F, Podda M, Bacciu D, Micheli A (2020) A fair comparison of graph neural networks for graph classification. In: *International conference on learning representations*
- Fey M, Lenssen JE (2019) Fast graph representation learning with PyTorch geometric. In: *ICLR workshop on representation learning on graphs and manifolds*
- Fukushima K (1980) Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol Cybern* 36(4):193–202. <https://doi.org/10.1007/BF00344251>
- Gama F, Marques AG, Leus G, Ribeiro A (2019) Convolutional neural network architectures for signals supported on graphs. *IEEE Trans Signal Process* 67(4):1034–1049. <https://doi.org/10.1109/TSP.2018.2887403>
- Gao H, Ji S (2019) Graph U-Nets. In: *International Conference on Machine Learning*, pp 2083–2092, ISSN: 1938-7228 Section: Machine Learning
- Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for Quantum chemistry. In: *Proceedings of the 34th international conference on machine learning, vol 70, JMLR.org, Sydney, NSW, Australia, ICML'17*, pp 1263–1272
- Goodfellow I, Bengio Y, Courville A (2016) *Deep learning*. MIT Press, Cambridge
- Gori M, Monfardini G, Scarselli F (2005) A new model for learning in graph domains. In: *Proceedings of the 2005 IEEE international joint conference on neural networks, vol 2*, pp 729–734. <https://doi.org/10.1109/IJCNN.2005.1555942>, ISSN: 2161-4407

- Hagberg AA, Schult DA, Swart PJ (2008) Exploring network structure, dynamics, and function using networkx. In: Varoquaux G, Vaught T, Millman J (eds) Proceedings of the 7th Python in Science Conference, Pasadena, CA USA, pp 11–15
- Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st international conference on neural information processing systems, Curran Associates Inc., Long Beach, California, USA, NIPS'17, pp 1025–1035
- Hammond DK, Vandergheynst P, Gribonval R (2011) Wavelets on graphs via spectral graph theory. *Appl Comput Harmonic Anal* 30(2):129–150. <https://doi.org/10.1016/j.acha.2010.04.005>
- Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput* 20(1):359–392. <https://doi.org/10.1137/S1064827595287997>
- Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th international conference on learning representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings, OpenReview.net
- Knyazev B, Taylor GW, Amer M (2019) Understanding attention and generalization in graph neural networks. In: Wallach H, Larochelle H, Beygelzimer A, Alché-Buc F, Fox E, Garnett R (eds) Advances in neural information processing systems 32, Curran Associates, Inc., pp 4202–4212
- Kriege NM, Johansson FD, Morris C (2020) A survey on graph kernels. *Appl Netw Sci* 5(1):6. <https://doi.org/10.1007/s41109-019-0195-3>
- Le Gall F (2014) Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th international symposium on symbolic and algebraic computation, pp 296–303
- LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. *Neural Comput* 1(4):541–551. <https://doi.org/10.1162/neco.1989.1.4.541>
- Lee J, Lee I, Kang J (2019) Self-attention graph pooling. In: International conference on machine learning, pp 3734–3743, ISSN: 1938-7228 Section: Machine Learning
- Li M, Ma Z, Wang YG, Zhuang X (2020) Fast Haar transforms for graph neural networks. *Neural Netw* 128:188–198. <https://doi.org/10.1016/j.neunet.2020.04.028>
- Li Q, Han Z, Wu XM (2018) Deeper insights into graph convolutional networks for semi-supervised learning. In: McIlraith SA, Weinberger KQ (eds) Proceedings of the thirty-second AAAI conference on artificial intelligence, (AAAI-18), the 30th innovative applications of artificial intelligence (IAAI-18), and the 8th AAAI symposium on educational advances in artificial intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, AAAI Press, pp 3538–3545
- Li Y, Tarlow D, Brockschmidt M, Zemel RS (2016) Gated graph sequence neural networks. In: Proceedings of the 4th international conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings
- Luzhnica E, Day B, Lio' P (2019) Clique pooling for graph classification. [arXiv:1904.00374](https://arxiv.org/abs/1904.00374)
- Ma Y, Wang S, Aggarwal CC, Tang J (2019) Graph convolutional networks with EigenPooling. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining, Association for Computing Machinery, Anchorage, AK, USA, KDD '19, pp 723–731, <https://doi.org/10.1145/3292500.3330982>
- Micheli A (2009) Neural network for graphs: a contextual constructive approach. *IEEE Trans Neural Netw* 20(3):498–511. <https://doi.org/10.1109/TNN.2008.2010350>
- Monti F, Boscaini D, Masci J, Rodola E, Svoboda J, Bronstein MM (2017) Geometric deep learning on graphs and manifolds using mixture model CNNs. pp 5115–5124
- Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M (2019) Weisfeiler and leman go neural: higher-order graph neural networks. *Proc AAAI Conf Artif Intel* 33(01):4602–4609. <https://doi.org/10.1609/aaai.v33i01.33014602>
- Morris C, Kriege NM, Bause F, Kersting K, Mutzel P, Neumann M (2020) TUDataset: a collection of benchmark datasets for learning with graphs. In: ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020), [arXiv:2007.08663](https://arxiv.org/abs/2007.08663)
- Ng AY, Jordan MI, Weiss Y (2002) On spectral clustering: analysis and an algorithm. In: Dietterich TG, Becker S, Ghahramani Z (eds) Advances in neural information processing systems, vol 14. MIT Press, Cambridge, pp 849–856
- Poulin V, Théberge F (2019) Ensemble clustering for graphs. In: Aiello LM, Cherifi C, Cherifi H, Lambiotte R, Lió P, Rocha LM (eds) Complex networks and their applications VII, Studies in Computational Intelligence. Springer, Cham, pp 231–243, https://doi.org/10.1007/978-3-030-05411-3_19

- Ranjan E, Sanyal S, Talukdar P (2020) ASAP: adaptive structure aware pooling for learning hierarchical graph representations. *Proc AAAI Conf Artif Intel* 34(04):5470–5477. <https://doi.org/10.1609/aaai.v34i04.5997>
- RAPIDS Development Team (2018) RAPIDS: collection of libraries for end to end GPU data science
- Scarselli F, Yong SL, Gori M, Hagenbuchner M, Tsoi AC, Maggini M (2005) Graph neural networks for ranking Web pages. In: *The 2005 IEEE/WIC/ACM international conference on web intelligence (WI'05)*, pp 666–672. <https://doi.org/10.1109/WI.2005.67>
- Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. *IEEE Trans Neural Netw* 20(1):61–80. <https://doi.org/10.1109/TNN.2008.2005605>
- Schomburg I, Chang A, Ebeling C, Gremse M, Heldt C, Huhn G, Schomburg D (2004) BRENDA, the enzyme database: updates and major new developments. *Nucl Acids Res*. <https://doi.org/10.1093/nar/gkh081>
- Shervashidze N, Schweitzer P, Leeuwen EJV, Mehlhorn K, Borgwardt KM (2011) Weisfeiler-Lehman graph kernels. *J Mach Learn Res* 12:2539–2561
- Simonovsky M, Komodakis N (2017) Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp 3693–3702
- Tomita E, Tanaka A, Takahashi H (2006) The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor Comput Sci* 363(1):28–42. <https://doi.org/10.1016/j.tcs.2006.06.015>
- Traag VA, Waltman L, van Eck NJ (2019) From Louvain to Leiden: guaranteeing well-connected communities. *Sci Rep* 9(1):5233. <https://doi.org/10.1038/s41598-019-41695-z>
- Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: *International conference on learning representations*
- Vinyals O, Bengio S, Kudlur M (2016) Order matters: sequence to sequence for sets. In: *Proceedings of the 4th international conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings*
- Vishwanathan SVN, Schraudolph NN, Kondor R, Borgwardt KM (2010) Graph Kernels. *J Mach Learn Res* 11(40):1201–1242
- Wale N, Watson IA, Karypis G (2008) Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl Inform Syst* 14(3):347–375. <https://doi.org/10.1007/s10115-007-0103-5>
- Wang Y, Li M, Ma Z, Montufar G, Zhuang X, Fan Y (2020) Haar Graph Pooling. *Proc Int Conf Mach Learn* 1
- Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K, Jegelka S (2018) Representation learning on graphs with jumping knowledge networks. In: *International conference on machine learning, PMLR*, pp 5453–5462, ISSN: 2640-3498
- Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? In: *International conference on learning representations*
- Yanardag P, Vishwanathan S (2015) Deep graph kernels. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, association for computing machinery, New York, NY, USA, KDD '15*, pp 1365–1374. <https://doi.org/10.1145/2783258.2783417>
- Ying R, You J, Morris C, Ren X, Hamilton WL, Leskovec J (2018) Hierarchical graph representation learning with differentiable pooling. In: *Proceedings of the 32nd international conference on neural information processing systems, Curran Associates Inc., Montréal, Canada, NIPS'18*, pp 4805–4815
- Yuan H, Ji S (2019) StructPool: structured graph pooling via conditional random fields
- Zhang L, Wang X, Li H, Zhu G, Shen P, Li P, Lu X, Shah SAA, Bennamoun M (2020) Structure-feature based graph self-adaptive pooling. In: *Proceedings of the web conference 2020, Association for Computing Machinery, New York, NY, USA, WWW '20*, pp 3098–3104. <https://doi.org/10.1145/3366423.3380083>
- Zhang M, Cui Z, Neumann M, Chen Y (2018) An end-to-end deep learning architecture for graph classification. In: *Proceedings of the thirty-second AAAI conference on artificial intelligence, (AAAI-18), the 30th innovative applications of artificial intelligence (IAAI-18), and the 8th AAAI symposium on educational advances in artificial intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018*, pp 4438–4445