K-Plex Cover Pooling for Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

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 pattern. 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 molecular and social graph classification shows that KPLEXPOOL achieves state-of-the-art performances, supporting the intuition that well-founded graph-theoretic approaches can be effectively integrated in learning models for graphs.

1 Introduction and Related Works

Graph neural networks 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 neural processing of graphs typically leverages message passing between neighboring nodes [2] 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 [14], but previously known as contextual structure processing [18, 1]. 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 realize 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. The definition of a robust, general, and efficient graph pooling mechanism is made difficult by the irregular nature of the data and by the lack of a reference ordering of nodes between samples. Approaches in literature are addressing the problem from a (more or less explicit) community discovery perspective by considering node connectivity patterns (topological approaches) [22, 17] or by aggregating the nodes based on their similarity in the neural embedding space (adaptive approaches) [28, 15, 3]. Our work proposes a novel link between pooling operators and two consolidated concepts in graph theory and combinatorial algorithms, namely, k-plexes and graph covers. The former provides 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 relates 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. We show how KPLEXPOOL can be effectively integrated into deep graph networks, with excellent performances that generalize well to structures of different nature and topology. Our contribution is not just a straightforward application of k-plexes to deep learning for graphs. Rather, KPLEXPOOL is the result of the careful design and integration of different graph enumeration and 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 [16] 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 at the cross-roads between graph algorithmics and machine learning, as for the Weisfeiler-Lehman graph kernel case [21]. Overall, 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

Preliminaries 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 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 $\phi : V \to \mathbb{R}^{h_V}$ and $\psi : V \times V \to \mathbb{R}^{h_E}$ 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 $\psi(e) = \mathbf{0}$.

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. 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 $\cup_{S \in \mathcal{S}} S = V$.

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

$$V' = V(G') = \{v'_1, \ldots, v'_c\}, \quad \text{and} \quad E' = E(G') = \big\{\{v'_i, v'_j\} \mid E\big(G[S_i, S_j]\big) \neq \emptyset\big\},$$

where $E\big(G[S_i, S_j]\big) = E(G) \cap (S_i \times S_j)$. 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 $\phi' : V' \to \mathbb{R}^{h_V}$ and $\psi' : V' \times V' \to \mathbb{R}^{h_E}$ 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:

$$\phi'(v'_i) = \beta\left(\{\phi(v) \mid v \in S_i\}\right), \quad \text{and} \quad \psi'\big(\{v'_i, v'_j\}\big) = \gamma\left(\{\psi(e) \mid e \in E\big(G[S_i, S_j]\big)\}\right),$$

where β and γ are arbitrary aggregation functions defined over multisets of feature vectors [26]. Differently from other partitioning-based graph coarsening methods [7, 19, 17, 24], in our approach a node may belong to multiple k-plexes. This is also a key difference between CLIQUEPOOL [16] 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.

***k*-Plex cover algorithm** We propose an algorithm, whose pseudocode is shown in Algorithms 1 and 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 [5, 6], 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).

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 higher priority $f(v)$. The node v will be then used as a starting node for retrieving the next k-plex $S \subseteq V$ by Algorithm 2 (S will eventually include v). The

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:     $v \leftarrow \mathrm{argmax}_{u \in U} f(u)$
5:     $S \leftarrow$ 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:     $u \leftarrow \mathrm{argmax}_{w \in C}\, g(w)$
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 $g$.
```

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 a successive execution of Algorithm 2 may contain previously removed nodes. The algorithm stops when eventually all the nodes are removed from U, and then returns 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

$$\forall u \in S: \ |S \setminus N(u)| \leq k\ , \tag{1}$$

$$\forall u \in S: \ |S \setminus N(u)| = k \implies C \setminus N(u) = \emptyset \tag{2}$$

where Equation (1) states that every node u in S needs to have at least $|S| - k$ adjacent nodes in S and Equation (2) 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 Equation (1) 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 Equation (1) by selecting a node u from C according to priority g. It needs to preserve Equation (2) 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.

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 $O(\sum_v d(v)) = O(m)$ time. The time cost of Algorithm 1 is bounded by $O(mn)$, as it performs $O(n)$ updates of S, U, and f, and $O(n)$ calls to Algorithm 2, which requires $O(m)$ time.

3 Experiments and Discussion

Experimental setting We tested KPLEXPOOL against related methods from literature on four molecular graph datasets, namely DD [8], NCI-1 [23], ENZYMES [20], and PROTEINS [4], and five social network datasets, COLLAB, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, and REDDIT-MULTI-5K [27]. All datasets have been retrieved from the TU-Dortmund collection [13].

For fairness, each pooling method has been tested by plugging it into the same standardized architecture (BASELINE), comprising $\ell \in \{2, 3\}$ convolutional blocks, followed by two dense layers, the latter interleaved by dropout with probability 0.3. Every convolutional block is formed by two GNN layers (GCN [14] or GRAPHSAGE [12]) with Jumping Knowledge [25, CONCAT variant] 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 $h \in \{64, 128\}$ units and a ReLU activation function [11]. For every other model, a pooling

Table 1: Test classification accuracy on chemical benchmarks ($\mathrm{mean} \pm \mathrm{std}$) and their average ranks. Bold highlights the best performing model. OOR stands for *out of resources* (results in italic are reported from [9] in similar but not fully equivalent conditions).

	DD	ENZYMES	NCI-1	PROTEINS	Avg. Rank
BASELINE	74.79 ± 3.08	43.00 ± 10.82	78.08 ± 2.38	71.61 ± 5.35	3.50
GRACLUS	77.42 ± 3.45	42.67 ± 7.82	78.06 ± 2.39	74.12 ± 3.36	2.75
TOPKPOOL	73.35 ± 4.29	39.17 ± 9.79	74.09 ± 7.29	74.12 ± 4.05	5.00
SAGPOOL	74.75 ± 3.10	37.67 ± 10.23	78.01 ± 1.68	73.31 ± 4.54	5.00
DIFFPOOL	OOR (*74.96 ± 3.5*)	$\mathbf{46.00 \pm 9.17}$	76.76 ± 2.37	75.02 ± 4.14	2.75
KPLEXPOOL	$\mathbf{77.76 \pm 2.92}$	39.67 ± 7.52	$\mathbf{79.17 \pm 1.73}$	$\mathbf{75.11 \pm 2.80}$	$\mathbf{1.75}$

Table 2: Test classification accuracy on social network benchmarks ($\mathrm{mean} \pm \mathrm{std}$) and their average ranks. Bold highlights the best performing model. OOR stands for *out of resources* (results in italic are reported from [9] in similar but not fully equivalent conditions).

	COLLAB	IMDB-B	IMDB-M	REDDIT-B	REDDIT-5K	Avg. Rank
BASELINE	74.44 ± 2.55	69.20 ± 6.92	$\mathbf{47.20 \pm 4.27}$	84.85 ± 6.70	51.71 ± 3.12	$\mathbf{2.40}$
GRACLUS	72.80 ± 1.10	68.70 ± 4.45	$\mathbf{47.20 \pm 2.93}$	86.85 ± 2.84	53.77 ± 1.29	2.80
TOPKPOOL	73.30 ± 2.96	68.20 ± 7.81	46.93 ± 3.03	78.60 ± 4.30	50.33 ± 2.84	4.80
SAGPOOL	73.40 ± 2.47	65.40 ± 6.28	46.33 ± 3.68	80.15 ± 7.49	49.79 ± 2.63	5.20
DIFFPOOL	70.92 ± 2.95	68.80 ± 8.04	47.07 ± 1.73	OOR (*89.08 ± 1.6*)	OOR (*53.78 ± 1.4*)	2.80
KPLEXPOOL	$\mathbf{76.20 \pm 2.08}$	$\mathbf{72.00 \pm 4.78}$	46.60 ± 4.08	86.45 ± 3.50	50.65 ± 3.41	2.80

layer is placed after the first $\ell - 1$ convolutional blocks and its output feed to the next block. For KPLEXPOOL we set $k \in \{1, 2, 4, 8\}$, while we used $r \in \{1/4, 1/2, 3/4\}$ as node-reduction factor for TOPK-, SAG-, and DIFFPOOL. For KPLEXPOOL and GRACLUS we used both *sum* and *max* (concatenated) as node aggregation (β) and *sum* for edge aggregation (γ). All models have been implemented using PyG [10]. Our experimental approach followed the standardized reproducible setting in [9] (stratified 10-fold with an inner grid-search on a 10% validation split per fold).

Discussion Tables 1 and 2 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. 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 [9]. On REDDIT-5K, only the topological pooling of GRACLUS yields to higher accuracy. 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, 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 the last column of Tables 1 and 2, where we report the average rank of each model separately on chemical and social datasets. We also report that KPLEXPOOL has the *best overall average rank* at 2.33, followed by GRACLUS on par with DIFFPOOL at 2.78.

KPLEXPOOL has state-of-the-art performances in 5 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.

References

[1] Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual Graph Markov Model: A Deep and Generative Approach to Graph Processing. In *International Conference on Machine Learning*, pages 294–303, July 2018. ISSN: 1938-7228 Section: Machine Learning.

[2] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, September 2020. ISSN 0893-6080.

[3] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *Proceedings of the 37th International Conference on Machine learning (ICML)*, 2020.

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

[5] Alessio Conte, Roberto Grossi, and Andrea Marino. Clique covering of large real-world networks. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 1134–1139, Pisa, Italy, April 2016. Association for Computing Machinery. ISBN 978-1-4503-3739-7.

[6] Alessio Conte, Roberto Grossi, and Andrea Marino. Large-scale clique cover of real-world networks. *Information and Computation*, 270:104464, February 2020. ISSN 0890-5401.

[7] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29 (11):1944–1957, November 2007. ISSN 0162-8828.

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

[9] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A Fair Comparison of Graph Neural Networks for Graph Classification. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HygDF6NFPB`.

[10] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 1025–1035, Long Beach, California, USA, December 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

[13] Kristian Kersting, Nils M Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL `graphkernels.cs.tu-dortmund.de`.

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

[15] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling. In *International Conference on Machine Learning*, pages 3734–3743, May 2019. ISSN: 1938-7228 Section: Machine Learning.

[16] Enxhell Luzhnica, Ben Day, and Pietro Lio'. Clique pooling for graph classification. *arXiv:1904.00374 [cs, stat]*, April 2019. arXiv: 1904.00374.

[17] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. Graph Convolutional Networks with EigenPooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 723–731, Anchorage, AK, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6201-6.

[18] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[19] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[20] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

[21] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(null):2539–2561, November 2011.

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

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

[24] Yuguang Wang, Ming Li, Zheng Ma, Guido Montufar, Xiaosheng Zhuang, and Yanan Fan. Haar Graph Pooling. *Proceedings of the International Conference on Machine Learning*, 1, 2020.

[25] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5449–5458. PMLR, 2018.

[26] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

[27] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[28] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 4805–4815, Montréal, Canada, December 2018. Curran Associates Inc.