Revisiting Virtual Nodes in Graph Neural Networks for Link Prediction

Anonymous Author(s)
Affiliation
Address
email

Abstract

It is well known that the graph classification performance of graph neural networks often improves by adding an artificial virtual node to the graphs, which is connected to all nodes in the graph. Intuitively, the virtual node provides a shortcut for message passing between nodes along the graph edges. Surprisingly, the impact of virtual nodes with other problems is still an open research question.

In this paper, we adapt the concept of virtual nodes to the link prediction scenario, where we usually have much larger, often dense, and more heterogeneous graphs. In particular, we use multiple virtual nodes per graph and graph-based clustering to determine the connections to the graph nodes. We also investigate alternative clustering approaches (e.g., random or more advanced) and compare to the original model with a single virtual node. We conducted extensive experiments over different datasets of the Open Graph Benchmark (OGB) and analyze the results in detail. We show that our virtual node extensions yield rather stable performance increases and allow standard graph neural networks to compete with complex state-of-the-art models, as well as with the models leading the OGB leaderboards.

1 Introduction

Link prediction is an important task to complete graphs that are missing edges in various domains: citation networks [15], social networks [3], medical drug interaction graphs [1], or knowledge graphs (KGs) [13]. Numerous kinds of models have been proposed to solve the link prediction problem, ranging from KG-specific predictors [13] to graph neural networks [15, 43]. Note that, over dense biomedical interaction networks, the latter turned out to work especially well [11].

In this work, we focus on *graph neural networks* (GNNs) for link prediction. Many of the popular GNNs are based on the message-passing scheme, which computes node embeddings based on iteratively aggregating the features of (usually direct/one-hop) neighbor nodes along the graph edges [9]. Interestingly, best performance is usually obtained by only considering two to three hops of neighbors (i.e., 2-3 layers in the GNN). One main reason identified for this is *over-smoothing*, the problem that node representations become indistinguishable when the number of layers increases [23]. Recently, [4] further demonstrate the so-called *over-squashing* phenomenon, the problem that with long-range dependencies information from the exponentially-growing receptive field is compressed into fixed-length node vectors, and thus cannot be propagated successfully. While it is likely that link prediction most often depends on the local node neighborhood, it is not beyond imagination that there are critical long-range dependencies (e.g., complex chains of drug-drug or drug-protein interactions). Hence, using a small number of layers to overcome the above problems results in *under-reaching*.

There have been several recent proposals to overcome under-reaching. On the one hand, several works propose techniques that allow for larger numbers of GNN layers [38, 34, 25, 6, 32, 45, 20]. However, although [6] show that over-smoothing happens particularly in dense graphs, the link prediction

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

experiments in these works consider citation or recommendation networks, but not the especially dense biomedical ones. And our experiments over the latter suggest that the reported results are not generalizable to the more challenging biomedical data. On the other hand, there are approaches that adapt the message-passing scheme to consider neighbors beyond the one-hop neighborhood: based on graph diffusion [5, 17, 2, 36, 28, 18] and other theories [30, 42]. However, most of these models are relatively complex and, in fact, in our experiments over the challenging graphs from the Open Graph Benchmark (OGB) [11], several ran out of memory. Moreover, the majority has not considered link prediction, while this problem was recently shown to be more difficult than node classification [44].

In this paper, we propose a simple but elegant solution to under-reaching based on the concept of *virtual nodes* [9, 21, 31, 12]. Virtual nodes are well known to often improve the graph classification performance of graph neural networks, where an artificial virtual node is added to every graph and connected to all nodes in the graph. While the virtual nodes were originally thought as representations of the entire graph, they also provide shortcuts for message passing between nodes along the graph edges. Surprisingly, the impact of virtual nodes for the link prediction problem has not been investigated yet. The reason for this might be that the often very large and heterogeneous "network" graphs in link prediction are of very different nature and require novel/adapted solutions (e.g., a protein interaction network may easily contain millions of nodes, whereas a molecule to be classified contains usually less than fifty).

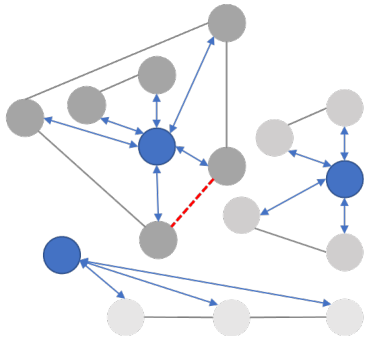


Figure 1: We cluster graph nodes belonging together (gray shades) and connect them to a common virtual node (blue). This eases information exchange and hence prediction of links between nodes not directly related (red).

We extend the virtual node concept to link prediction:

- *We propose to use multiple virtual nodes in the link prediction scenario and describe a graph-based technique to connect them to the graph nodes.* Consider Figure 1. In a nutshell, we use a graph clustering algorithm to determine groups of nodes in the graph that belong together and then connect these nodes to a common virtual node. In this way, under-reaching is decreased because clustered nodes can share information easily; at the same time, the nodes are spared of unnecessary information from unrelated nodes (i.e., in contrast to the single virtual node model).
- We also investigate alternative methods to determine the virtual node connections (e.g., random or more advanced) and compare to the original model with a single virtual node.
- We conducted extensive experiments over different kinds of datasets of the OGB, provide ablation studies that confirm the superiority of our proposed techniques, and analyze the results in detail.
- Most importantly, we show that our virtual node extensions yield rather stable performance increases and allow standard graph neural networks to compete with complex state-of-the-art models that also try to improve message passing, as well as with the models leading the OGB leaderboards.

2 Related Work

In the following, we give an overview on approaches that are similar from a technical perspective; for a more detailed summary, see Appendix A. For a more general overview of the large and diverse field of link prediction models, we refer the reader to recent works that provide good summaries [29, 44].

Deeper GNNs. Several techniques address over-smoothing and hence allow for constructing deeper GNNs to solve under-reaching. These models range from the simple but efficient message propagation in SGC [34, 25] and APPNP [17] and connections in JKNet [38], to more advanced proposals [6, 32, 45, 20] such as the differentiable aggregation functions in DeeperGCN [20]. However, although [6] show that over-smoothing happens particularly in dense graphs, the experiments in most of these works consider citation or recommendation networks, but not the especially dense and important biomedical ones. And our experiments over the latter suggest that the reported results are not generalizable to the more challenging biomedical data.

Beyond One-Hop Neighbors. Recently, graph diffusion methods are used in various ways to determine the message targets and thus extend standard message passing beyond the one-hop neigh-

borhood. [5] use k-hop random walks to extend the node features. APPNP [17] applies personalized PageRank to propagate the node predictions generated by a neural network. Other models concatenate [2] or aggregate [36, 28, 40] node embeddings in every layer using a diffusion-based transition matrix. The diffusion-based graph neural network (GDC) [18] aggregates information from multiple neighborhood hops at each layer by sparsifying a generalized form of graph diffusion. Subsequent works use diffusion methods on multiple scales [24, 27, 35] and integrate attention [33].

[30] take higher-order graph structures at multiple scales into account during message passing based on the k-dimensional Weisfeiler and Leman graph algorithm. The model closest to our approach is P-GNN [42]. It assigns nodes to random clusters ("anchor-sets") and then creates a message for each node for every anchor-set, while ignoring the message passing from original direct neighbors. Our virtual nodes represent an alternative means to aggregate messages from multiple graph nodes which are not necessarily direct neighbors. We also explore the idea of similar random assignments in our context, but show that more elaborate techniques generally work better. Observe that the above approaches are all relatively complex. We considered several of them in our experiments and many terminated with memory errors. Moreover, only few of them have been evaluated for link prediction.

Virtual Nodes. To the best of our knowledge, virtual nodes have only been considered in the context of graph classification so far, where a single virtual node (also called *supernode*) is added to the graph to be classified and connected to all graph nodes [9, 21, 31, 12]. Note that the original idea was to compute a graph embedding in parallel with the node embeddings and even connected the virtual node only in one direction (i.e, via edges from the graph nodes) instead of bidirectionally [21]. Despite it is a well known trick, the advantage of using virtual nodes has never been fully understood. In this work, we focus on link prediction and considerably extend the virtual node techniques. In addition, we analyze theoretically and empirically how the latter improve GNN performance.

3 Preliminaries

Link Prediction. We consider an undirected *graph* $G = (V, E)$ with *nodes* V and *edges* $E \subseteq V \times V$. Note that this basic choice is only for ease of presentation. All our techniques work for directed graphs and, with simple adaptation, also for graphs with labelled edges. We assume V to be ordered and may refer to a node by its index in V. For a node $v \in V$, $\mathcal{N}_v$ denotes the set of its neighbors. Given two nodes, the *link prediction task* is to predict whether there is a link between them.

Message-Passing Graph Neural Networks. In this paper, we usually use the term *graph neural networks* (GNNs) to denote GNNs that use message passing as described in [9]. These networks compute for every $v \in V$ a node representation h_v^ℓ at layer $\ell \in [1, 2, \ldots, k]$, by *aggregating* its neighbor nodes based on a generic aggregation function and then *combine* the obtained vector with $h_v^{\ell-1}$ as below; h_v^0 are the initial node features.

$$h_v^\ell = \text{COMBINE}^\ell\Big(h_v^{\ell-1},\ \text{AGGREGATE}^\ell\big(\{h_u^{\ell-1} \mid u \in \mathcal{N}_v\}\big)\Big) \tag{1}$$

Link prediction with GNNs is usually done by combining (e.g., concatenating) the final representations h_u^L, h_v^L, of the nodes u, v under consideration and passing them through several feed-forward layers with a final sigmoid function for scoring. We follow this approach.

We further use $[1, n]$ to denote an interval [1,2,..., n]. Note that we do not define virtual nodes on purpose here, because we extend the concept considerably in this paper.

4 Virtual Nodes in Graph Neural Networks for Link Prediction

So far, virtual nodes have been only used for graph classification. Link prediction scenarios are different in that the graphs are usually very large, heterogeneous, sometimes dense, and the task is to predict a relationship that might strongly influenced depend on surrounding relations. In the following, we propose approaches that fit these scenarios.

4.1 Multiple Virtual Nodes

Our main goal of using virtual nodes is to provide a shortcut for sharing information between the graph nodes. However, the amount of information in a graph with possibly millions of nodes is enormous,

and likely too much to be captured in a single virtual node embedding. Further, not all information is equally relevant to all nodes. Therefore we suggest to use *multiple virtual nodes* $S = \{s_1, s_2 \ldots, s_n\}$[1] each being connected to a subset of graph nodes, as determined by an assignment $\sigma : V \to [1, n]$; n is considered as hyperparameter. We propose different methods to obtain σ:

Random (GNN-RM). Most simple, we can determine a fixed σ randomly once with initialization.

Increased Randomness (GNN-RM^F). Similarly a random assignment, but initialized with every forward pass. In this way, a single bad assignment does not determine the overall performance.

Clustering (GNN-CM). Many types of graph data incorporate a certain cluster structure (e.g., collaboration or social networks) that reflects which nodes belong closely together. We propose to connect such nodes in a cluster to a common virtual node, such that the structure inherent to the given graph is reflected in our virtual node assignment σ. More precisely, during initialization, we use a generic clustering algorithm which, given a number m, creates a set $C = \{C_1, C_2 \ldots, C_m\}$ of clusters (i.e., sets of graph nodes) by computing an assignment $\rho : V \to [1, m]$, assigning each graph node to a cluster. We then obtain σ by choosing $m = n$ and $\sigma = \rho$.

In this work, we decided for the METIS clustering [14] which turned out to provide a good trade off between quality and efficiency. Nevertheless, our idea is generic and can be applied with arbitrary algorithms. We will show ablation experiments for alternatives (e.g., Graclus [8] and Diffpool [41]).

Advanced Clustering (GNN-CM$^+$). Not every type of graph data contains an inherent cluster structure or one that is sufficiently expressed, though. Furthermore, using a fixed clustering, we obtain a deterministic algorithm again taking the risk that we completely rely on a single, possibly not ideal, virtual node assignment – there may be critical long range dependencies that go beyond clusters. For these cases, we propose an alternative approach, which breaks up the determinism by extending the above clustering approach as follows. We choose a relatively large m, with $m \gg n$, and apply the clustering algorithm as above, during initialization. Then, with every epoch, we randomly guess an assignment $\sigma' : [1, m] \to [1, n]$ of clusters to virtual nodes. We then define $\sigma(n) := \sigma'(\rho(n))$. Note that this approach is inspired by [7] where a similar technique is applied to create batches based on clusters. Further note that we determine σ' with every epoch instead of every forward pass since the the computation takes quite some time on large datasets and we observed that this yields good results.

4.2 The Model

We integrate the multiple virtual nodes into a generic message-passing graph neural network by extending the approach from [11] to the setting with multiple virtual nodes by computing node representations h_v^ℓ for a node $v \in V$ at layer ℓ as follows:

$$h_{s_i}^\ell = \text{COMBINE}_{VN}^\ell\Big(h_{s_i}^{\ell-1},\ \text{AGGREGATE}_{VN}^\ell\big(\{h_u^{\ell-1} \mid u \in V, \sigma(u) = i\}\big)\Big) \tag{2}$$

$$h_v^\ell = \text{COMBINE}^\ell\Big(h_v^{\ell-1} + h_{s_{\sigma(v)}}^\ell,\ \text{AGGREGATE}^\ell\big(\{h_u^{\ell-1} \mid u \in \mathcal{N}_v\}\big)\Big) \tag{3}$$

Note that the highlighted adaptation of the standard GNN processing from Equation (1) is only minor – but powerful. In our implementation, COMBINE_{VN}^ℓ is addition combined with linear layers and layer normalization, and we use sum for $\text{AGGREGATE}_{VN}^\ell$.

4.3 Analysis: Virtual Nodes Change Influence

Influence Score. Following [38, 17], we measure the sensitivity (also, *influence*) of a node x to a node y by the *influence score* $I(x, y) = e^{\mathrm{T}} \frac{\partial h_x^k}{\partial h_y^0} e$; e is a vector of all ones, h_x^k is the embedding of x at the k^{th} layer, see Equations (1) and (3). For a k-layer GNN, the influence score is known to be proportional in expectation to the k-step random walk distribution from x to y (see Theorem 1, [38]):[2]

$$\mathbb{E}[I(x, y)] \propto P_{rw}(x \to y, k) = \sum_{r \in R^k} \prod_{\ell=1}^{k} \frac{1}{deg(v_r^\ell)}, \tag{4}$$

[1] Since notation V is standard for nodes, we use S for the set of virtual nodes. Think of "supernodes".

[2] Note that the theorem makes some simplifying assumptions (e.g., on the shape of GNN), see [38] for details.

$(v_r^0, v_r^1, ..., v_r^k)$ are the nodes in the path r from $x := v_r^0$ to $y := v_r^k$, R^k is the set of paths of length k. In what follows, we will exploit this relationship and argument in terms of the probability P_{rw}.

Virtual Nodes. For simplicity, consider the influence score in an m-regular graph; there we have $P_{rw}(x \to y, k) = \frac{|R^k|}{m^k}$. We hypothesize that we can come to similar conclusions in a general graph with average degree m. Consider the message passing between two distant nodes x and y. (I) In case the shortest path from x to y is of length $> k$, a k-layer GNN cannot capture it, and the probability $P_{rw}(x \to y, k)$ is obviously zero. If we then consider virtual nodes in the GNN layer (even with only one), we can pass messages from x to y through the virtual nodes and obtain a nonzero probability. (II) Consider the case where there is a shortest path of length $\leq k$ between x and y. By adding a virtual node s in one GNN layer, the probability changes to:

$$P_{rw}^s(x \to y, k) = P_{rw}(x \to y, k) + P_{rw}(x \to s, s \to y) = \frac{|R^k|}{(m+1)^k} + \frac{1}{(m+1)|V|}. \quad (5)$$

Compared to the original probability, we get the following impact ratio for using virtual nodes:

$$ir = \frac{m^k}{(m+1)^k} + \frac{m^k}{(m+1)|V||R^k|}. \quad (6)$$

When m is large enough, ir can be approximated by $ir \simeq \left(1 + \frac{m^{k-1}}{|V||R^k|}\right)$. Here, we see that the impact of virtual nodes grows when m increases. Our experiments confirm this theoretical observation.

Multiple Virtual Nodes. In view of multiple virtual nodes, the above analysis gets even more appealing. We continue along these lines and assume there is a shortest path of length $\leq k$ between x and y. If x and y connect to the same virtual node s, then Equation (5) changes as follows:

$$P_{rw}^s(x \to y, k) = \frac{|R^k|}{(m+1)^k} + \frac{1}{(m+1)|C_s|}. \quad (7)$$

Since the set C_s of nodes connecting to s is much smaller than V, the impact of multiple virtual nodes is greater than that of a single virtual node.

Note that, if x and y do not connect to the same virtual node, the probability just slightly decreases from $\frac{|R^k|}{m^k}$ to $\frac{|R^k|}{(m+1)^k}$.

In Appendix B, we further show that using multiple virtual nodes is related to the labeling trick [44] and distance encoding [22], which theoretically improve the expressivness in learning structural link representations.

5 Evaluation

We conducted extensive experiments to empirically investigate the following questions:

1. How does the existing approach with **one virtual node perform in link prediction**?
2. **Do multiple virtual nodes improve performance**, how do our proposed approaches compare?
3. In particular, **are approaches based on the graph structure better**?
4. How exactly do virtual nodes support link prediction? **When do they help particularly?**

5.1 Datasets

To answer the above questions, we focused on challenging data from the OGB: `ddi`, a drug-drug interaction network; `ppa10`, a subset of the protein-protein association network `ppa` containing only 10% of the train edges (but full valid/test); and `collab`, an author collaboration network. Since the datasets are not only very different in type but also in various other critical graph parameters and this is reflected in the performance of the models, we show relevant statistics from [11] in Table 1.[3] The datasets vary strongly in size with `ddi` < `collab` < `ppa10`. Yet, `ddi` is very dense. The clustering coefficient intuitively reflects the "cliquishness" of the graph's subgraphs. The large graph diameters suggest that the data suits testing under-reaching. Appendix C provides further details.

[3] See Tables 2 and 3 in [11]. Note that we computed the numbers for `ppa10` (which we focus on due to a lack of resources) using the same techniques as [11].

Table 1: Overview of link prediction datasets from the Open Graph Benchmark. All graphs are undirected, do not have edge features, and all but `ddi` have node features.

	#Nodes	#Edges	Average Node Deg.	Average Clust. Coeff.	MaxSCC Ratio	Graph Diameter
`ddi`	4,267	1,334,889	500.5	0.514	1.000	5
`ppa10`	509,860	11,217,535	8.3	0.019	0.983	29
`ppa`	576,289	30,326,273	73.7	0.223	0.999	29
`collab`	235,868	1,285,465	8.2	0.729	0.987	23

Table 2: Overall results over different types of graph data; **top**: state of the art (two best per dataset) according to OGB leaderboard, leaderboard results are marked by * and averages over 10 runs; **second**: models with similar goal to our approach ("-": ran out of memory); **bottom**: GCN, SAGE, and GIN with different configurations of virtual nodes; (second) best results per part are (light) gray, overall best **bold**, second best underlined.

	`ddi` Hits@20	`ppa10` Hits@100	`collab` Hits@50
SAGE +dist.*	0.8239 ± 0.0437	n/a	n/a
DEA+JKNet*	0.7672 ± 0.0265	n/a	n/a
Adamic Adar*	0.1861 ± 0.0000	n/a	**0.6417 ± 0.0000**
SEAL*	0.3056 ± 0.0386	n/a	0.6364 ± 0.0071
DeeperGCN*	n/a	n/a	0.6187 ± 0.0045
GCN+JKNet*	0.6056 ± 0.0869	n/a	n/a
SGC	0.0676 ± 0.0586	0.0814 ± 0.0013	0.4635 ± 0.0197
P-GNN	0.1050 ± 0.0000	-	-
APPNP	0.1492 ± 0.0298	0.0572 ± 0.0160	0.3185 ± 0.0205
GCN-GDC	0.2550 ± 0.1242	-	-
SAGE-GDC	0.3141 ± 0.1254	-	-
GIN-GDC	0.2180 ± 0.0786	-	-
GCN*	0.3707 ± 0.0507	n/a	0.4714 ± 0.0145
GCN	0.5062 ± 0.2186	0.1313 ± 0.0084	0.4955 ± 0.0064
- VN	0.5932 ± 0.2390	0.1258 ± 0.0082	0.5049 ± 0.0088
- RM	0.5580 ± 0.1852	0.1205 ± 0.0059	0.5083 ± 0.0109
- RM^F	0.5874 ± 0.1104	0.1116 ± 0.0094	0.5046 ± 0.0049
- CM	0.6322 ± 0.1565	0.1299 ± 0.0050	0.5181 ± 0.0076
- CM$^+$	0.6554 ± 0.1543	0.1399 ± 0.0071	0.5128 ± 0.0129
SAGE*	0.5390 ± 0.0474	n/a	0.5463 ± 0.0112
SAGE	0.6128 ± 0.2122	0.1024 ± 0.0050	0.5662 ± 0.0149
- VN	0.7160 ± 0.1457	0.0853 ± 0.0154	0.5875 ± 0.0091
- RM	0.6627 ± 0.0414	0.1131 ± 0.0039	0.5830 ± 0.0087
- RM^F	0.7299 ± 0.1543	0.1105 ± 0.0023	0.6067 ± 0.0063
- CM	**0.8819 ± 0.0341**	0.1077 ± 0.0150	0.6056 ± 0.0105
- CM$^+$	0.8041 ± 0.1058	0.0963 ± 0.0099	0.5940 ± 0.0262
GIN	0.4829 ± 0.1608	0.1139 ± 0.0058	0.5768 ± 0.0179
- VN	0.6523 ± 0.0446	0.1316 ± 0.0049	0.5863 ± 0.0254
- RM	0.6388 ± 0.2294	0.1337 ± 0.0045	0.5412 ± 0.0174
- RM^F	0.5363 ± 0.1028	0.1269 ± 0.0026	0.5335 ± 0.0087
- CM	0.6544 ± 0.0960	0.1349 ± 0.0034	0.5821 ± 0.0081
- CM$^+$	0.5962 ± 0.1575	**0.1591 ± 0.0069**	0.5557 ± 0.0026

5.2 Baselines

For a competitive comparison, we considered important baselines (see descriptions in Section 2):

- The deep GNNs **SGC**, **APPNP**, **DeeperGCN**, and two variants of **JKNet**.
- Approaches extending message passing: **P-GNN**, **APPNP**, **GCN-GDC**, **SAGE-GDC**, **GIN-GDC**.
- The popular GNNs **GCN** [16], **SAGE** [10], and **GIN** [37], which we then extend with multiple virtual nodes, and also the versions with a single virtual node, **GCN-VN**, **SAGE-VN**, and **GIN-VN**.
- Finally, we report the results of the two best models per dataset according to the OGB leaderboard: **SAGE +dist.**, an extension of SAGE which extends the final vector to be scored by a distance encoding [19]; **DEA+JKNet**, an extension of JKNet that applies distance encodings during message aggregation and uses statistics as additional node features [39]; **Adamic Adar**, a custom link prediction heuristic [3]; and **SEAL**, which applies a GNN on subgraphs around link source and target after re-labelling the nodes in the subgraphs based on their distances to the latter nodes [43].

Observe that for many of the above models, the results on the OGB link prediction datasets are unknown. Hence our study also delivers important comparison numbers for the community.

5.3 Results

We provide extensive experimental results, detailed ablation experiments, and an analysis in the subsequent section. Some further results (e.g., on the full `ppa` dataset) are given in Appendix E.

Overall Results, Table 2.
1. Impact of Virtual Nodes. We first compare to the GNNs GCN, SAGE, and GIN. The existing approach of using a single virtual node (GNN-VN) yields good improvements over `ddi`, slight improvements over `collab`, but no definitive ones over `ppa10`. The numbers for GNN-RM and GNN-RM^F reflect the randomness of their connections to the multiple virtual nodes, there is no clear trend. Nevertheless, they clearly outperform the original models, with only few exceptions. The increased randomness by re-assigning the virtual nodes with every forward pass (GNN-RM^F) seems to suit SAGE but not the others. *Multiple virtual nodes turn out to be an efficient means to boost the link prediction performance of GNNs* if they are applied correctly. Our virtual node connections based on the graph structure (i.e., clustering with GNN-CM) yield consistently good improvements over `ddi` and `collab`, and help even sligthly on the challenging `ppa10` dataset (GCN represents an exception there). Observe that GNN-CM and GNN-CM$^+$ are not always the best ones, but yield reliably good results, in contrast to the other models with virtual nodes (see variability of gray shades). Interestingly, the advanced clustering yields especially good performance over the latter, while its results on the other datasets are not convincing. Generally, the improvements of the virtual node models are strongest on `ddi`. For an in-depth result analysis see Section 5.4.
2. Comparison to Related Works. Both the deep GNNs as well as the models that use complex message-passing techniques perform disappointing and, overall, much worse than the standard GNNs. We did hyperparameter tuning also for these models and it is hard to explain these numbers. However, most of the original evaluations focus on node or graph classification and consider very different types of data – often the standard citation networks [26] and, in fact, on `collab` we see the best numbers. For a more detailed discussion regarding P-GNN see Appendix F. Even if we assume that these numbers can be improved, the models do not seem apt for link prediction; in particular, the complex ones: many do not run at all on realistic link prediction data but yield memory errors.
3. Comparison to OGB Leaderboard. Most importantly, *our virtual node extensions make standard GNNs outperform the models on the leaderboard*. In particular, their performance is much more stable. The results of the best models from the leaderboard vary strongly with the different datasets, or have not been provided all. None of these models can be called "good" overall, given the numbers in the rest of the table; in fact, SEAL and Adamic Adar perform rather bad on the very dense `ddi`.

Impact of Virtual Nodes on Number of GNN Layers and Efficiency, Figure 2. On `ddi`, we see much more variety than over `collab`. Interestingly, the best numbers of layers are unusually high: 6 layers for GCN, and also for GCN-VN on `collab`. We observe that the virtual node(s) allow both GCN-VN and GCN-CM to reach their best score on `ddi` at 10 layers, which is remarkable for that very dense dataset (i.e, prone to over-smoothing). This is similar for GCN-CM model on `collab`. The figure also gives an idea of the runtime increase with using virtual nodes. It compares the 6-layer models, and shows the 4-layer GCN-CM which obtains similar performance as 6-layer GCN(-VN).

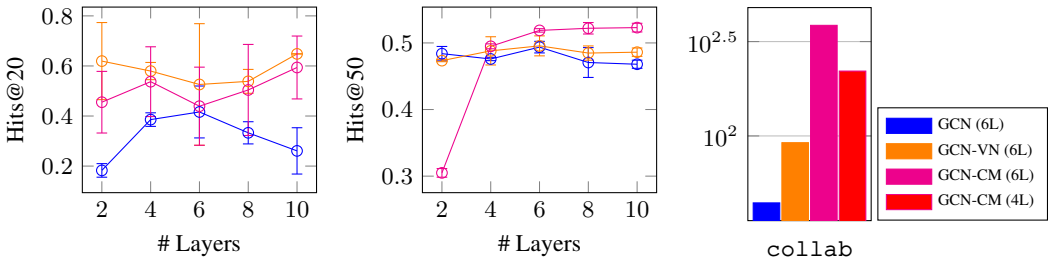


Figure 2: Performance depending on layers: Hits@k and time per epoch (sec.); `ddi` (left), `collab`.

Table 3: Comparison of using the virtual nodes at every and only at the last layer; Hits@20, `ddi`.

	GCN	SAGE	GIN
	0.5062 ± 0.2186	0.6128 ± 0.2122	0.4829 ± 0.1608
- VN	0.5932 ± 0.2390	$\underline{0.7160 \pm 0.1457}$	$\underline{0.6523 \pm 0.0446}$
- VN_{OL}	0.6180 ± 0.0088	0.5167 ± 0.1364	0.6472 ± 0.0542
- CM	$\underline{0.6322 \pm 0.1565}$	$\mathbf{0.8819 \pm 0.0341}$	$\mathbf{0.6544 \pm 0.0960}$
- CM_{OL}	$\mathbf{0.6338 \pm 0.1188}$	0.6151 ± 0.1545	0.4420 ± 0.1694

Using Virtual Nodes Only at the Last GNN Layer, Table 3. In [4], it is shown that using a fully connected adjacency matrix at the last layer of a standard GNN helps to overcome over-squashing. We investigated if it is better to use the virtual nodes only at the last layer. However, we see that this can lead to extreme drops in performance (with SAGE and GIN), while there are only minor improvements with GCN. Interestingly, the trend is similar comparing GNN-VN to GNN-VN_{OL} and GNN-CM to GNN-CM_{OL}. We conclude that, in case over-squashing effects occur, the GNN-VN/-CM architectures solve them better than simpler solutions, and our architecture is better overall.

Impact of Virtual Node Number, Figure 3. First, consider the configurations of the best models for the overall results in Table 2, which are provided in Table 5 in the appendix. Here, we see that the chosen numbers of virtual nodes are indeed random for the "random" models, but GNN-CM consistently uses a high number of virtual nodes, which also suits it better according to our theoretical analysis in Section 4.3. The more detailed analysis varying the numbers of virtual nodes does not show a strong sensitivity or particular trend regarding this parameter. Yet, the best models use relatively high numbers (i.e., ≥ 8) and also have smallest standard deviations with those. Furthermore, GCN-CM tends to use more than SAGE-CM. Note that there is a trade off between number of virtual nodes and intra-cluster test edges, which we discuss in Section 5.4.

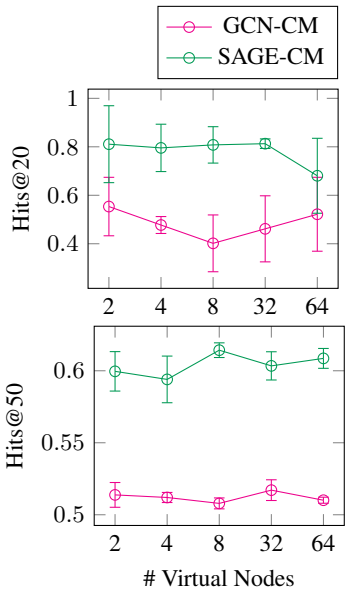


Figure 3: Impact of virtual node number; `ddi` (top) and `collab`.

Impact of Clustering Algorithm, Table 4. Our architecture is generic in the clustering algorithm, and we here give an idea about the effects of varying that. Graclus is similar in nature to METIS in that it also creates partitions based on the adjacency matrix, but it took much longer to run. Diffpool considers the node features and yields improvements for GCN, but we were not able to run it with the larger datasets. Over the `ddi` data, there is no clear winner and, given it's efficiency, METIS turns out to be a good solution.

5.4 Discussion

Our experimental results show that our approach with multiple virtual nodes based on graph-based clustering yields performance increases for various GNNs and, in contrast to all other models, rather

Table 4: Comparison of clustering algorithms to determine virtual node connections; Hits@20, ddi.

	GCN	SAGE	GIN
CM_{metis}	0.6322 ± 0.1565	$\mathbf{0.8819 \pm 0.0341}$	$\underline{0.6544 \pm 0.0960}$
CM^{+}_{metis}	$\underline{0.6554 \pm 0.1543}$	$\underline{0.8041 \pm 0.1058}$	0.5962 ± 0.1575
$\text{CM}_{graclus}$	0.6324 ± 0.1048	0.5406 ± 0.3769	$\mathbf{0.7299 \pm 0.0604}$
$\text{CM}_{diffpool}$	$\mathbf{0.7392 \pm 0.0381}$	0.7556 ± 0.1252	0.5436 ± 0.1572

stable performance across the datasets. Yet, there are strong differences based on the properties of the data (see Table 1), which are also reflected in the best model configurations (see Appendix D).

Dense Graphs with Medium/High Clustering Coefficient. Over ddi, we see strongest improvements for all virtual-node models, because virtual nodes help to overcome common issues of GNNs with this kind of data: (1) dense data is particularly prone to lead to over-smoothing [6]; (2) over-squashing is likely to happen as well. To see the latter, consider an exponentially-growing neighborhood where each node has 500 neighbors on average. Of course, not all the neighbors will be different, but it is still likely that an embedding of a standard size will not capture all neighbors successfully if the number of GNN layers gets larger. This is reflected in the results: our best GCN and SAGE model both use only 2 layers (GIN uses more but performs worse). In contrast, many of the virtual-node models use more layers successfully. To see the reasons for the that, recall Equation (6), where we show that a very large m increases the impact of the virtual node(s), and thus decreases the negative impact of the (too) many other neighbors. Furthermore, the empirical results confirm our proposed theory regarding multiple virtual nodes (see Equation (7)). We see particularly good numbers for GNN-CM, which exploits the clustering inherent in the given graph. GNN-CM^{+}, which considers this given clustering only on a lower level, is shown to perform worse than GNN-CM overall. In fact, we computed the percentage of test edges that occur in the "virtual node cluster" (see Table 6 in the appendix) and it shows that the numbers for the advanced clustering are very similar to the random one, meaning the randomly merged smaller clusters break the data's structure too much.

Graphs with Large Problem Radius and Low Clustering Coefficient. Over ppa10, all GNNs use an unusually high number of layers, which hints at a large problem radius (e.g., GCN, which performs especially well, uses 7 layers). Given the very low clustering of the data in addition, ppa10 represents a special challenge. With the multiple virtual nodes, GNN-CM performs again better than GNN-VN. On the other hand, it does not perform much better than the random models on data without cluster structure. This can be explained by its choice of number of virtual nodes, which is consistently high, but then there are less test edges within a virtual node cluster (see appendix Table 6). We hence see here that the positive effect of having many virtual nodes (recall Equation (7)) cancel out the benefits of clustering. Our advanced clustering, which merges some local clustering with randomness, is able to achieve best results with GCN and GIN (with SAGE, all models perform rather bad over ppa10). This can be explained by the fact that it randomly merges some local clusters – with each epoch anew – and hence allows more messages to pass across "virtual node clusters".

Graphs with Large Problem Radius and High Clustering Coefficient.

Interestingly, the problem radius of collab seems to be equally large. However, here we have clustering and, accordingly, GNN-CM is the only model which performs very good for all GCN, SAGE, and GIN. The trends in the models' performance and the corresponding explanations are similar to those for ddi but much less expressed due to the much smaller average node degree.

6 Conclusions

In this paper, we propose a simple but elegant graph neural network extension using multiple virtual nodes which considerably increases their link prediction performance. We also advance research by providing an in-depth study of virtual nodes for link prediction. Our work provides fundamental results of very technical nature, without direct negative societal or ethical impacts. Nevertheless, graph data has a pervasive impact in many diverse fields and the popularity of graph neural networks certainly increases the danger of an application with negative effects (even if due to ignorance). We believe that biomedicine is one of the application areas where our research provides the greatest benefits, and we hope that our papers motivate other researchers to target similar data/applications.

References

[1] Khushnood Abbas, Alireza Abbasi, Shi Dong, Ling Niu, Laihang Yu, Bolun Chen, Shi-Min Cai, and Qambar Hasan. Application of network link prediction in drug discovery. *BMC bioinformatics*, 22(1):1–21, 2021.

[2] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proc. of ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 21–29. PMLR, 2019.

[3] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[4] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *Proc. of ICLR*, 2021.

[5] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Proc. of NIPS*, volume 29. Curran Associates, Inc., 2016.

[6] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proc. of ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 2020.

[7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proc. of KDD*, pages 257–266. ACM, 2019.

[8] 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, 2007.

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proc. of ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[10] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Proc. of NIPS*, pages 1024–1034, 2017.

[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Proc. of NeurIPS*, 2020.

[12] Katsuhiko Ishiguro, Shin-ichi Maeda, and Masanori Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks. *CoRR*, abs/1902.01020, 2019.

[13] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[14] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998.

[15] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*. OpenReview.net, 2017.

[17] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proc. of ICLR*. OpenReview.net, 2019.

[18] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Proc. of NeurIPS*, pages 13333–13345, 2019.

[19] Boling Li, Yingce Xia, Shufang Xie, Lijun Wu, and Tao Qin. Distance-enhanced graph neural network for link prediction.

[20] Guohao Li, Chenxin Xiong, Ali K. Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *CoRR*, abs/2006.07739, 2020.

[21] Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery. *CoRR*, abs/1709.03741, 2017.

[22] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Proc. of NeurIPS*, 33, 2020.

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proc. of AAAI*, volume 32, 2018.

[24] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *Proc. of ICLR*. OpenReview.net, 2019.

[25] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *Proc. of KDD*, pages 338–348. ACM, 2020.

[26] Qing Lu and Lise Getoor. Link-based classification. In *Proc. of ICML*, 2003.

[27] Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proc. of NeurIPS*, volume 32. Curran Associates, Inc., 2019.

[28] Zheng Ma, Junyu Xuan, Yu Guang Wang, Ming Li, and Pietro Liò. Path integral based convolution and pooling for graph neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Proc. of NeurIPS*, 2020.

[29] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33, 2016.

[30] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proc. of AAAI*, pages 4602–4609. AAAI Press, 2019.

[31] Trang Pham, Truyen Tran, Khanh Hoa Dam, and Svetha Venkatesh. Graph classification via deep learning with virtual nodes. *CoRR*, abs/1708.04357, 2017.

[32] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Adagcn: Adaboosting graph convolutional networks into deep models. 2021.

[33] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Direct multi-hop attention based graph neural network. *CoRR*, abs/2009.14332, 2020.

[34] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proc. of ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR, 2019.

[35] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. In Hal Daumé III and Aarti Singh, editors, *Proc. of ICML*, volume 119, pages 10432–10441. PMLR, 2020.

[36] Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, and Xueqi Cheng. Graph convolutional networks using heat kernel for semi-supervised learning. In Sarit Kraus, editor, *Proc. of IJCAI*, pages 1928–1934. ijcai.org, 2019.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proc. of ICLR*. OpenReview.net, 2019.

[38] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proc. of ICML*, pages 5453–5462. PMLR, 2018.

[39] Yichen Yang, Lingjue Xie, and Fangchen Li. Global and local context-aware graph convolution networks.

[40] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In Yike Guo and Faisal Farooq, editors, *Proc. of KDD*, pages 974–983. ACM, 2018.

[41] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Proc. of NeurIPS*, pages 4805–4815, 2018.

[42] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proc. of ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 7134–7143. PMLR, 2019.

[43] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proc. of NIPS*, pages 5171–5181, 2018.

[44] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Revisiting graph neural networks for link prediction. *arXiv preprint arXiv:2010.16103*, 2020.

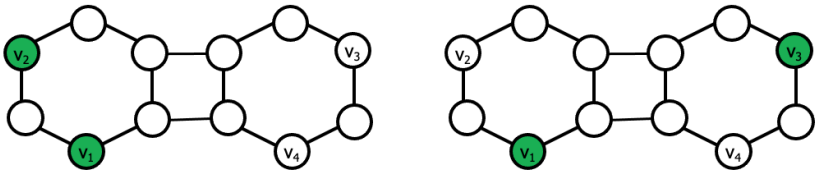
[45] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Proc. of NeurIPS*, 2020.

Checklist

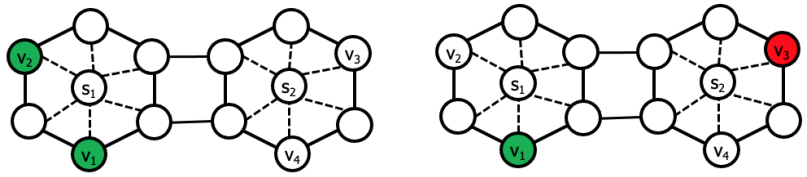
1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplementary.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We did too many experiments to have a detailed overview of all; we are happy to provide specific details upon reviewer request.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] We mainly use the OGB as well as other popular packages such as PyTorch which are well-known to be usable.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In supplementary.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]



(a) Without Virtual Nodes



(b) With 2 Virtual Nodes

Figure 4: Example of using two virtual nodes to improve expressiveness. In Figure 4(a), a typical message-passing GNN (see Section 3) computes the same representation for v_2 and v_3, and we cannot discriminate the node pairs (v_1, v_2) and (v_1, v_3) if we just use the node representation for link prediction, as it is usual. In Figure 4(b), by adding two virtual nodes s_1 and s_2, v_2 and v_3 have clearly different features (because they connect to different virtual nodes with different labels) and different representations after GNN embedding, so (v_1, v_2) and (v_1, v_3) can be easily discriminated.

A Additional Details on Related Works

Deeper GNNs. We mention simpler approaches in the Section 2. More advanced proposals are, for example, based on special features and connections [6], community-based normalization of node representations using random clustering [45], boosting techniques [32], or differentiable aggregation functions in DeeperGCN [20].

Beyond One-Hop Neighbors. Graph diffusion methods (i.e., in graph theory, techniques for spreading information between nodes) are used in various ways to determine the message targets and thus extend standard message passing beyond the *one-hop* neighborhood. [5] use k-hop random walks to aggregate node features and extend the latter by the aggregated ones. APPNP [17] applies personalized PageRank to propagate the node predictions generated by a neural network. Other models aggregate node embeddings in every layer, GraphHeat [36] using the heat kernel, PAN [28] the transition matrix of maximal entropy random walks, and PinSage [40] using random walks. [2] propose to concatenate embeddings aggregated using the transition matrices of k-hop random walks before applying one-hop neighbor aggregation. The diffusion-based graph neural network (GDC) [18] aggregates information from multiple neighborhood hops at each layer by sparsifying a generalized form of graph diffusion. Subsequent works use diffusion methods on multiple scales [24, 27, 35]. Recently, [33] integrated attention with diffusion-based message propagation.

B Additional Theoretical Results

Adding structure-related features such as a distance encoding [22] has been demonstrated to make graph representation learning more powerful. For link prediction, [44] propose the *labeling trick* extending distance encoding and making GNNs learn better link representations.

We first recall the definitions from [44] introducing the concept of labeling trick. Consider an undirected graph G as described in Section 3. In addition, the tensor $\mathbf{A} \in \mathbb{R}^{n \times n \times k}$ contains all node and edge features (if available). The diagonal components $\mathbf{A}_{v,v,:}$ denote the node features, while the off-diagonal components $\mathbf{A}_{u,v,:}$ denote the edge features of edge (u, v). The labeling trick uses a target node set $S \subseteq V$ and a labeling function to label all nodes in the node set V and stack the labels

with $\mathbf{A}$. A valid labeling trick must meet two conditions: (1) the nodes in S have different labels from the rest of the nodes, (2) the labeling function must be permutation invariant.

Let us recall our method using multiple virtual nodes. Assume we have multiple virtual nodes $S = \{s_1, ..., s_m\}$. $\forall u \in V$, we have the additional features for the node $l(u|S) = (h(s_1), ..., h(s_m))^{\mathrm{T}}(\gamma(u|s_1), ..., \gamma(u|s_m))$, where $\gamma(u|s_i) = 1$ if u is connected to the virtual node s_i, and $\gamma(u|v_i) = 0$ otherwise. $h(s_i)$ is the node representation of virtual node s_i, and is initialized by one-hot vectors so that each virtual node has different labels.

Our labeling strategy is not a valid labeling trick by the definition of [44]. First, S is not a subset of V, and we use addition instead of concatenation. Even if we extend V to $V \cup S$, our labeling strategy still does not fit the permutation-invariant requirement. Nevertheless, it can achieve similar effects in learning structural link representations.

Theorem 1. *In any non-attributed graphs with n nodes, if the degree of each node in the graph is between 1 and $\mathcal{O}(\log^{\frac{1-\epsilon}{2h}}(n))$ for any constant $\epsilon > 0$, given m virtual nodes which evenly divide the node set into m clusters, then there exists $\omega\left((m-1)^2(\frac{n^\epsilon}{m}-1)^3\right)$ many pairs of non-isomorphic links $(u, w), (v, w)$, such that an h-lalyer 1-WL-GNN (see definition in [22] and [44], one well-known example is GIN[37]) gives u, v the same representation, while using m virtual nodes can give u, v different representations.*

Proof. The proof can be separated into two steps. The first step is to prove that there exists $n/o(n^{1-\epsilon}) = \omega(n^\epsilon)$ many nodes that are locally h-isomorphic. This step is same as the proof of Theorem 2 in [44], so we omit the details here. After getting these locally isomorphic nodes, we denote the set of these nodes as V_{iso}. The second step is to find the non-isomorphic links.

Step 2. Let us partition $V_{iso} = \cup_{i=1} V_i$ where V_i is the subset of nodes connected to virtual node s_i. To be simple, we call each V_i a cluster, and the sizes of different clusters are assumed to be the same $|V_i| = |V_{iso}|/m$. Consider two nodes $u \in V_i$ and $v \in V_j$ from different clusters. Since both of them are in V_{iso}, so they have identical h-hop neighborhood structures, and h-layer 1-WL-GNN will give them the same representations. Then let us select another node w in V_i, h-layer 1-WL-GNN will also make (u, w) and (v, w) have the same representation.

However, if we use virtual nodes to label nodes and give them additional features, because u, w are in the same cluster while v, w belong to different clusters, (u, w) will have different representation from (v, w). Now let us count the number of such non-isomorphic link pairs Y, we can have:

$$Y \geq \prod_{i,j=1,j\neq i}^{m} |V_i||V_i - 1||V_j| = \frac{1}{2}m(m-1)\left(\left(\frac{|V_{iso}|}{m} - 1\right)\left(\frac{|V_{iso}|}{m}\right)^2\right)$$

Taking $|V_{iso}| = \omega(n^\epsilon)$ into the above in-equation, we get

$$Y \geq \frac{1}{2}m(m-1)\omega\left((\frac{n^\epsilon}{m} - 1)^3\right) = \omega\left((m-1)^2(\frac{n^\epsilon}{m} - 1)^3\right)$$

Example (power of using multiple virtual nodes). In Figure4, we show two cases with and without virtual nodes. Consider the nodes v_2, v_3 with the same local structure, which means they can get the same node representations by using 1-WL-GNN. So we cannot discriminate the links (v_1, v_2) and (v_1, v_3) if we just use 1-WL-GNN and concatenate the node representations for link prediction. However, if we add 2 virtual nodes and add extra features to each node. v_1 and v_2 get a new feature $(1, 0)$, v_3 get new feature $(0, 1)$. So it is easy to see (v_1, v_2) and (v_1, v_3) now have different representations.

C Additional Details on the Data

The datasets vary strongly in the number of nodes and edges with `ddi` < `collab` < `ppa10` < `ppa`. Yet, `ddi` shows a challenging average node degree and hence is very dense. The average clustering coefficient intuitively reflects the extent to which direct neighbors of a node are themselves direct neighbors, and thus measures the "cliquishness" of the graph's subgraphs. Here, we have `ppa10` < `ppa` < `ddi` < `collab`. Finally, the graph diameter varies similarly, and again shows that `ppa`

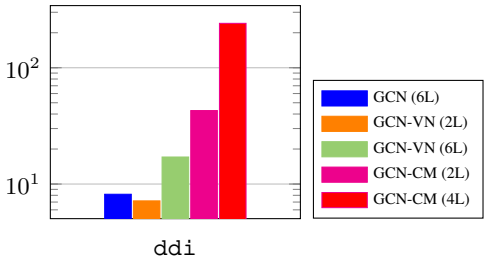


Figure 5: Performance depending on layers: time per epoch (sec.); `ddi`. We observe from Figure 2 that we get better Hits@20 for the virtual nodes models already at 2 layers than for GCN at 6 layers (its best score), hence we also compare these models. Here we see that a single virtual node can have a positive impact at the same time on both prediction scores and efficiency. The clustering takes more time.

represents a special challenge. The diameter does not necessarily reflect the problem depth directly, but it shows which maximal problem depth can be expected. In particular, the depths here hint at much larger numbers of GNN layers than the 2-3 layers we usually use, hence the data might suit testing under-reaching.

D Model Configurations and Training

We trained all models for 80 runs using the Bayesian optimization provided by wandb[4] and the following hyperparameters.

hidden dimension	32, 64, 128, 256
learning rate	0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001
dropout	0, 0.3, 0.6
# of layers	1-7
# of virtual nodes (random)	1-10
# of virtual nodes	1,2,4,8,16,32,64
SGC - K	2-7
APPNP - α	0.05, 0.1, 0.2, 0.3
GNN-GDC - k	64, 128
GNN-GDC - α	0.05, 0.1, 0.2, 0.3

Note that we used less virtual nodes in the selection for the models (-RM, -RM^F) since especially -RM^F was very slow and preliminary results showed that larger numbers did not change the results greatly – probably due to the randomness. We used maximally 64 virtual nodes due to memory issues with larger numbers (e.g., 128), especially on the larger datasets. We report the specific numbers of GNN layers and virtual nodes used by the trained models from Table 2 in Table 5. For the first clustering in GNN-CM$^+$, we created 200 clusters on `ddi` and `collab`, and 1000 on `ppa10`.

We trained as suggested by the OGB (e.g., the splits, negative sampling) but used a batch size of 2^{12} and adapted the number of runs due to lack of resources; we used 3 for all experiments in Section 5. However, we ran several of our models for 10 runs as required for results on the OGB leaderboards and the numbers are comparable. After the 80 runs, we ran the models with the best 3 configurations for 3 runs and chose the best of these model as the final model (configuration).

We used 500 epochs with a patience of 30. Furthermore, for `collab`, we used the validation edges during testing (OGB contains both settings, with and without them).

[4] `https://wandb.ai/site`

Table 5: Numbers of GNN layers and virtual nodes used by the trained models from Table 2.

	ddi		ppa10		collab	
	#Lay	#VNs	#Lay	#VNs	#Lay	#VNs
GCN	2	0	7	0	2	0
- VN	2	1	7	1	6	1
- RM	1	7	7	2	7	7
- RM^F	2	9	6	9	3	2
- CM	4	64	6	64	6	32
- CM$^+$	3	64	5	32	7	8
SAGE	2	0	4	0	5	0
- VN	4	1	6	4	4	1
- RM	4	10	5	6	7	4
- RM^F	4	3	6	9	7	3
- CM	3	8	6	64	5	32
- CM$^+$	3	16	6	4	6	2
GIN	6	0	2	0	2	0
- VN	5	1	7	1	4	1
- RM	2	5	6	8	3	7
- RM^F	1	4	4	7	3	9
- CM	4	64	4	64	3	64
- CM$^+$	5	1	6	16	4	16

E Additional Experimental Results

Results on ppa

The ppa dataset is challenging in both its size and density. Since we were missing the resources to run experiments for all baselines on this dataset, we compare our best models (trained only on ppa10, we did not do additional hyperparameter tuning) to the OGB leaderboard in Table 7. Please note that these are intermediate results due to the limitation of time/resources, final results are expected to be better. However, we already see that our virtual node approach is able to improve the GNN results – even if only trained on 10% on the data.

Runtime

We show the runtimes on ddi in Figure 5. Here we see that a single virtual node can have a positive impact at the same time on both prediction scores and efficiency, while the clustering takes more time.

Cluster Analysis

We computed additional statistics about our "virtual node clusters" (i.e., a cluster represents a set of nodes connected to the same virtual node). Our hypotheses was that our proposed clustering based on the graph structure better reflects the distribution of test edges than, for example, random clustering. We report the results in Table 6. For the -RM^F and -CM$^+$ models we report two numbers. The upper one shows the average number of intra-cluster test edges over 10 runs. The numbers in the lower part distinguish the actual edges and reflect how many different test edges occur in a cluster over the 10 runs. These numbers hence represent lower and upper bounds respectively.

As expected, the numbers for -CM are in between those bounds. For ddi, we see that the -CM$^+$ and -RM^F numbers are very similar, while the ones for -CM$^+$ are much better over collab and ppa10.

F Details about P-GNN

The model closest to our approach is the position-aware graph neural network (P-GNN) [42]. It assigns nodes to random subsets of nodes called "anchor-sets", and then learns a non-linear aggregation

Table 6: Percentage of intra-cluster test edges, using numbers of virtual nodes between 8 and 64. For RM and CM$^+$, we average over 10 different seeds but drop standard deviation (which is negligible) for readability. The numbers in brackets with -CM$^+$ are the numbers of original METIS clusters which are then merged into "virtual node clusters"; we see no great sensitivity for them. We highlight our proposed graph-based clustering models we used in the experiments.

	ddi				
	4	8	16	32	64
CM	49.37	30.33	17.43	03.81	04.08
RM / RM^F	25.03	12.50	06.26	03.11	01.57
CM$^+$ (200)	25.31	12.95	06.82	03.61	02.13
RM^F	94.36	73.75	47.48	27.16	14.60
CM$^+$ (200)	94.65	73.51	47.85	27.06	14.89
CM$_{graclus}$	100.0	100.0	100.0	99.95	99.73

	collab				
	4	8	16	32	64
CM	81.01	75.85	68.29	59.92	52.10
RM / RM^F	25.05	12.50	06.22	03.14	01.52
CM$^+$ (200)	58.97	52.38	48.90	47.12	46.31
CM$^+$ (1000)	53.33	45.63	41.62	39.69	38.68
RM^F	94.35	73.76	47.51	27.21	14.25
CM$^+$ (100)	95.50	86.10	73.14	61.84	56.39
CM$^+$ (200)	96.68	85.83	71.55	59.88	53.01
CM$^+$ (1000)	96.42	82.39	67.36	54.70	47.09

	ppa10				
	4	8	16	32	64
CM	79.95	73.87	70.37	64.97	58.97
RM / RM^F	25.01	12.51	06.25	03.12	01.57
CM$^+$ (1000)	51.58	43.55	39.24	37.43	36.34
RM^F	94.39	73.69	47.51	27.18	14.61
CM$^+$ (200)	96.94	86.19	72.78	62.59	56.12
CM$^+$ (1000)	96.36	83.07	66.03	53.56	45.06

scheme that combines node feature information from each anchor-set and weighs it by the distance between the node and the anchor-set. That is, it creates a message for each node for every anchor-set, instead of for each direct neighbor.

We ran experiments with P-GNN but did not obtain conclusive results. It did not run on the larger datasets. For `ddi`, we considered the number of anchor nodes as hyperparameter since the fixed choice of 64 from the experiments of [42] did not yield good results. However, larger numbers such as 128 or 512 resulted in very large runtimes (9 hrs / epoch). The result in Table 2 is an intermediate best value after 50 runs. We contacted the authors and they indeed mentioned that the model is not very scalable and suggested to use just the anchor-set distance as additional features, instead of overtaking the adapted message passing as well. We did not do this extra experiment since the SAGE +dist model, whose numbers we report, follows a similar approach.

Table 7: Results on `ppa`: OGB leaderboard compared to our best models.

	Hits@100
SAGE +dist.*	n/a
DEA+JKNet*	n/a
Adamic Adar*	0.3245 ± 0.0000
SEAL*	0.4880 ± 0.0316
GCN*	0.1867 ± 0.0132
SAGE*	0.1655 ± 0.0240
GCN-CM$^+$	0.2089 ± 0.0079
GIN-CM$^+$	0.2399 ± 0.0091