SMGRL: Scalable Multi-resolution Graph Representation Learning

Anonymous authors
Paper under double-blind review

Abstract

Graph convolutional networks (GCNs) allow us to learn topologically-aware node embeddings, which can be useful for classification or link prediction. However, they are unable to capture long-range dependencies between nodes without adding additional layers—which in turn leads to over-smoothing and increased time and space complexity. Further, the complex dependencies between nodes make mini-batching challenging, limiting their applicability to large graphs. We propose a Scalable Multi-resolution Graph Representation Learning (SMGRL) framework that enables us to learn multi-resolution node embeddings efficiently. Our framework is model-agnostic and can be applied to any existing GCN model. We dramatically reduce training costs by training only on a reduced-dimension coarsening of the original graph, then exploit self-similarity to apply the resulting algorithm at multiple resolutions. The resulting multi-resolution embeddings can be aggregated to yield high-quality node embeddings that capture both long- and short-range dependencies. Our experiments show that this leads to improved classification accuracy, without incurring high computational costs.

1 Introduction

When working with graph-structured data, we often wish to learn latent vector representations for nodes within the graph—often referred to as node embeddings. These representations, which typically aim to capture the topological structure of the graph, can be used for tasks such as node classification and link prediction, or can be combined to obtain a representation of the entire graph. Message passing algorithms, where a node's embedding is updated based on its neighbors' embeddings, are a natural way to capture topological structure. Graph convolutional networks (GCNs, Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) learn the form of these updates using neural networks.[1] Since GCNs learn an algorithm to obtain embeddings (rather than directly learn a mapping from nodes to embeddings), we can apply the resulting algorithm to new or modified graphs, allowing GCNs to operate in an inductive fashion. This gives them a clear advantage over transductive algorithms such as node2vec (Grover & Leskovec, 2016) or DeepWalk (Perozzi et al., 2014), which learn a one-to-one mapping from node to embedding, often based on random walks over the graph, and cannot generalize to nodes not appearing in the training graph.

GCNs have achieved impressive performance on many graph-based classification, prediction, and simulation tasks (Klicpera et al., 2019; Brockschmidt, 2020; Bianchi et al., 2021). However, there are limits to their representational power. A single-layer GCN aggregates information only from a node's immediate neighborhood, ignoring any longer-range dependencies. This can be addressed by adding more layers—a K-layer GCN incorporates information from a node's K-hop neighborhood. For small values of K, this can improve representational power; however as the reach of the message passing algorithm expands, the inferred representations become increasingly homogeneous—a phenomenon known as oversmoothing (Li et al., 2016; Oono & Suzuki, 2019; Cai & Wang, 2020)—leading to a decrease in performance.

[1] The original GCN (Kipf & Welling, 2017) was so-named because it deploys a message-passing algorithm that approximates spectral convolution on the graph; in this work, we use the term more broadly to cover algorithms that learn a message-passing algorithm on a graph.

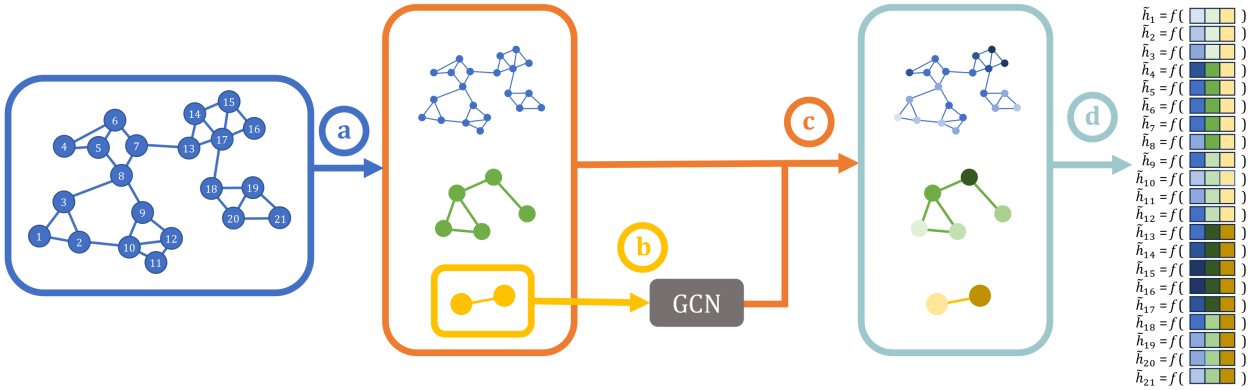


Figure 1: A schematic of SMGRL. a) We coarsen our original graph to obtain a hierarchy $\mathcal{G}_0, \ldots, \mathcal{G}_L$. b) We train our GCN on the coarsest graph, $\mathcal{G}_L$. c) We use the trained GCN to obtain embeddings at each layer of the hierarchy. d) We combine the per-layer embeddings to get an overall representation for each node.

In addition, the computational and memory requirements of GCNs scale poorly when compared with standard feedforward neural networks. The number of messages—and hence the computational cost—scales with the number of edges, which can grow quadratically in the numbere of nodes. Memory requirements are inflated due to the connectivity pattern of the graph: the representation of a node relies on all the nodes in its K-hop neighborhood. This means that, when minibatching, we must store in memory not just the set of nodes selected for the minibatch, but also their K-hop neighborhood (or an appropriately informative subset thereof).

One way to reduce the computational and memory requirements is to train the GCN on a smaller graph that is "similar" to the original graph. Loukas (2019) showed that a class of coarsening algorithms can be used to construct graphs that have similar spectral properties to the original graph. Huang et al. (2021) show that training a GCN on this coarsened graph can be seen as an approximation to training on the full graph, and that we can use the resulting algorithm on the original graph to obtain node embeddings. Moreover, theoretical and empirical results suggest that this procedure effectively acts as a regularizer, leading to improved generalization performance over the baseline GCN.

Since the GCN deployed on the coarsened graph is ultimately deployed on the original graph to obtain node embeddings, the approach of Huang et al. (2021) does not address the question of how to capture long-range dependencies without increasing the number of layers in the GCN (increasing computational cost and risking washing out the impact of local structure due to oversmoothing). Instead, we construct a hierarchy of graphs $\mathcal{G}_0, \ldots, \mathcal{G}_L$ by aggregating nodes at level ℓ into "supernodes" at level $\ell + 1$. The intuition here is that one-hop neighborhoods in the coarser-resolution graphs at the top of the hierarchy capture long-range behavior such as interactions between communities and cliques, while one-hop neighborhoods in the finer-resolution graphs at the bottom of the hierarchy capture local interactions. Rather than learn embeddings separately for each level of the hierarchy, we infer level-specific embeddings using a single GCN trained on the coarsest level. The resulting embeddings—each capturing information at a different length-scale—can be combined into a single, multi-resolution embedding for each node. This approach, which we summarize in Figure 1, is reminiscent of hierarchical methods that learn representations at multiple levels of a hierarchy (Chen et al., 2018; Akyildiz et al., 2020; Liang et al., 2021; Zhong et al., 2022; Guo et al., 2021; Jiang et al., 2020). However, unlike such methods, we do not incur high computational costs associated with training or refining at each level of the hierarchy.

We find that the resulting algorithm—whose computational and memory requirements are almost identical to those of the coarsened GCN approach of Huang et al. (2021)—typically outperforms both a GCN trained on the original graph, *and* the coarsened GCN algorithm, which only considers a single resolution. It also outperforms several existing hierarchical GCN algorithms, while having a much lower computational cost.

Our framework, which we denote Scalable, Multi-resolution Graph Representation Learning (SMGRL), consists of three components: a hierarchical graph coarsening algorithm (step a in Figure 1); a GCN learned on a coarsened graph (step b) and deployed at all levels of the hierarchy (step c); and a final aggregation step (step d). Each of these components is highly customizable. In particular, we explore three choices of GCN, and in the appendix explore the impact of different aggregation schemes.

2 Background

2.1 Graph convolutional networks

Representing nodes $v_i \in \mathcal{V}$ within a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in terms of embeddings $h_i \in \mathbb{R}^d$ allows us to distill important information about the graph topology in an easily digestible package. GCNs allow us to learn such embeddings in an inductive manner: rather than directly learn a mapping from $\mathcal{V}$ to $\mathbb{R}^d$, GCNs learn how to learn such embeddings. Concretely, they parametrize a message-passing algorithm that allows us to update node embeddings by passing messages along the graph's edges, and aggregating the messages arriving at each node.

Consider a node classification task where each node v_i has feature x_i and label y_i. Our goal is to find node embeddings h_i, and a corresponding classification function g_θ, that minimize some loss $\sum_{i=1}^{|\mathcal{V}|} \mathcal{L}(g_\theta(h_i), y_i)$. We do so by learning a message passing update rule of the form

$$h_i \leftarrow \textsc{Update}\left(x_i, \textsc{Aggregate}\left\{x_j : v_j \in \mathrm{Ne}(v_i; \mathcal{G})\right\}\right), \tag{1}$$

where $\mathrm{Ne}(v_i; \mathcal{G})$ indicates the neighborhood of v_i in $\mathcal{G}$. Multiple layers of the form in (1) can be stacked so that the embedding is based on a multi-hop neighborhood.

When $\textsc{Update}(m)$ takes the form $\sigma(Wm)$ and $\textsc{Aggregate}$ is an appropriately normalized summation, this can be seen as an approximation to graph convolution Kipf & Welling (2017). In this paper, we use the term graph convolutional network more broadly to refer to any message passing algorithm of the form of (1), allowing for more expressive aggregation methods (e.g., Xu et al., 2018; Veličković et al., 2018). We also include methods that use alternative definitions of neighborhoods Hamilton et al. (2017); Klicpera et al. (2019). By contrast, we use the terminology "graph neural network" to also include non-message-passing node and graph representation algorithms such as node2vec (Grover & Leskovec, 2016).

GCNs have proved a powerful tool for learning node embeddings. However, they do have a number of limitations. Firstly, GCNs typically have high time and space complexity. Training a GCN involves updating each node's embedding based on it's neighborhood once for each layer, meaning that the computational complexity of training a K-layer GCN scales as $O\left(K|\mathcal{E}|\right)$. If our training graph is dense, $|\mathcal{E}| \sim O\left(|\mathcal{V}|^2\right)$. Further, dependency between nodes makes minibatching challenging: if a GCN has K layers, then updating the embedding of a single node requires knowledge of the K-hop neighborhood of that node. This leads to increased memory requirements when minibatching, since we must augment a size-m minibatch with its K-hop neighborhood. In dense graphs, this can easily envelop a high proportion of the full graph.

Secondly, in a single-layer GCN the embedding of a node depends only on the node features of its immediate neighborhood. This limits our ability to learn long-range dependencies across the graph. To get around this, we can stack multiple layers, increasing the sphere of influence. However, adding more layers leads to over-smoothing—all embeddings become increasingly similar—which in turn leads to degradation in prediction performance (Li et al., 2016; Oono & Suzuki, 2019; Cai & Wang, 2020).

2.2 Graph coarsening and hierarchical embeddings

In many cases, graphs can be thought of as the realization of a latent hierarchical structure. For example, in a road network, we can think of a top-level graph connecting cities via major thoroughways, and a lower-level graph that also includes local roads. In a social network, we can think of individuals belonging to communities (such as schools, workplaces or hobbies), and consider the degree of connectivity between communities as well as the pattern of connectivity within communities. Such hierarchical representations allow us to

look at the graph at different resolutions, or length-scales: the original graph allows us to inspect local interactions (occurring on the length scale of one degree of separation between individuals), while the coarser representations allow us to consider long-range dependencies (where distance between nodes correspond to much higher average degrees of separation between individuals).

In general, an explicit hierarchy is not provided; however we can use various graph coarsening, node clustering or community detection algorithms to infer a hierarchy (e.g., Karypis & Kumar, 1998; Blondel et al., 2008; Loukas, 2019; Cai et al., 2021). Coarsening a graph involves constructing a sequence of graphs $\mathcal{G}_0, \ldots, \mathcal{G}_L$, such that $\mathcal{G}_0 := \mathcal{G}$ and nodes in $\mathcal{G}_\ell$ have a single parent in $\mathcal{G}_{\ell+1}$. We refer to the original graph $\mathcal{G}_0$ as the bottom layer of the hierarchy, and $\mathcal{G}_L$ as the top layer.

By learning embeddings at multiple layers in the hierarchy, we can capture variation at multiple levels of resolution. For example, we might jointly train GCNs at multiple levels in a hierarchy (Jiang et al., 2020; Guo et al., 2021), or learn a single GCN that spans all levels (Zhong et al., 2022). Such approaches have been found to increase expressive power over GCNs learned on the original graph. In addition, hierarchies have been successfully exploited to learn informative representations of entire graphs. For example, hierarchical pooling methods have been used to combine representations from multiple levels in a hierarchy, resulting in a multi-resolution graph embedding (Ying et al., 2018; Huang et al., 2019; Bandyopadhyay et al., 2020; Xin et al., 2021; Liu et al., 2021).

A related family of approaches use a hierarchical representation to refine embeddings (typically learned in a transductive manner). For example, we might learn embeddings at the coarsest layer $\mathcal{G}_L$, and then sequentially refine them to get embeddings at finer levels (Chen et al., 2018; Akyildiz et al., 2020; Liang et al., 2021). Or, we might start by learning embeddings on the original graph $\mathcal{G}_0$, coarsen these embeddings by propagating them up the hierarchy, and then refine them by propagating down the hierarchy (Hu et al., 2019; Li et al., 2020). While these approaches do not explicitly use multi-level representations in the final embedding, the refinement process biases the representations towards including information from the coarsened graphs.

An alternative use of graph coarsening is to reduce the computational cost of training a GCN. If our coarsened graph $\mathcal{G}_L$ is structurally similar to the original graph $\mathcal{G}_0$, we can train our GCN on $\mathcal{G}_L$ and then deploy it directly on $\mathcal{G}_0$. Coarsening algorithms that aim to maintain spectral properties of the original graph are an appropriate choice here (Loukas & Vandergheynst, 2018; Loukas, 2019; Bravo Hermsdorff & Gunderson, 2019; Cai et al., 2021). These representations are well-matched to the task of learning GCNs, which (for certain choices of Update and Aggregate in Equation 1) can be viewed as generalizations of approximate spectral convolution over the graph (Kipf & Welling, 2017). Indeed, Huang et al. (2021) show that a GCN trained on a spectrally coarsened graph can be seen as an approximation to a GCN trained on the full graph. In addition to reducing computation and memory requirements, training a GCN on a spectrally coarsened graph often leads to improved generalization, likely due to the approximation mechanism acting as a regularizer.

3 The proposed framework: SMGRL

Hierarchical node embeddings, such as those discussed in Section 2.2, allow us to aggregate information at multiple resolutions, incorporating both long- and short-range dependencies. The adjacency matrix of the original graph shows connectivity between individual nodes, while the adjacency matrices of the coarsened graphs show connectivity between clusters of nodes. However, jointly learning GCNs at multiple levels of granularity, or recursively refining embeddings at each level of a hierarchy, increases the computational cost over a single GCN learned on the original graph, making these approaches impractical for large graphs. To avoid this, we exploit the fact that graph coarsening algorithms such as those proposed by Loukas (2019) are explicitly designed to preserve spectral properties of the original graph. This suggests that a message passing algorithm learned at one level of granularity might be appropriate at multiple levels. This intuition is made concrete by the coarsened GCN framework of Huang et al. (2021), who train on the coarsest level of a spectrally-coarsened hierarchy, and use the resulting algorithm on the original graph.

We go further: we deploy a GCN learned on the coarsest graph at *all* levels of the hierarchy. This leads to a minimal increase in computational cost over Huang et al. (2021), but a potentially significant increase in representational power. If a GCN trained on level L is a good approximation to one trained on level 0,

then it should also be a good approximation at levels $0 < \ell < L$. The embeddings obtained at levels $\ell > 0$ will capture variation at longer length scales, without needing to add layers to our GCN. In particular, if the hierarchy obtained via the spectral coarsening algorithm aligns with intuitive notions of interconnected communities within the graph, embeddings at coarser resolutions will capture interactions between such communities.

This approach is reminiscent of the hierarchical GCNs described in Section 2.2, which combine embeddings at multiple resolutions. However, by reusing a GCN learned at the coarsest level, our computational costs are significantly lower than approaches that learn GCNs at all levels. Further, since all embeddings are obtained using a common aggregation rule, they can be linearly combined to obtain a single embedding for each node.

The computational cost of SMGRL scales similarly to that of the coarsened GCN algorithm of Huang et al. (2021); we provide a complexity analysis in Appendix A. The hierarchical coarsening can be pre-computed on CPU; further, since the algorithm is deterministic, we only need compute it once if we are using multiple random seeds or exploring multiple hyperparameter settings for our GCN.

Our framework consists of four steps: a) deploying a hierarchical coarsening algorithm; b) training a GCN on the coarsest graph; c) deploying this GCN on all levels of the hierarchy; and d) aggregating the resulting embeddings. This procedure is visualized in Figure 1.We provide details of each step below, and summarize the framework in Algorithm 1.

a) Construct a coarsened representation We begin by constructing hierarchically coarsened representations $\mathcal{G}_0, \ldots, \mathcal{G}_L$ of our graph (where $\mathcal{G}_\ell = (\mathcal{V}_\ell, \mathcal{E}_\ell)$ and $\mathcal{G}_0 := \mathcal{G}$), alongside associated coarsened features X_ℓ and labels Y_ℓ. Our goal is to obtain a hierarchy such that a GCN trained on one graph is appropriate for all graphs in the hierarchy. For this reason, we chose to use a graph coarsening method that (approximately) preserves spectral properties of the original graph.

Concretely, we follow the "variation neighborhoods" method introduced by Loukas (2019). Loosely, this algorithm constructs $\mathcal{G}_\ell$ from $\mathcal{G}_{\ell-1}$ by collapsing the neighborhood of a single node in $\mathcal{G}_{\ell-1}$ into a single node in $\mathcal{G}_\ell$. Neighborhoods to be collapsed are selected in order to minimize a spectral distance between the coarsened graph $\mathcal{G}_\ell$ and the previous level $\mathcal{G}_{\ell-1}$. The resulting coarsening operation is characterized by a projection matrix P_ℓ, which is also used to propagate features and labels up the hierarchy (e.g., $X_\ell = P_\ell X_{\ell-1}$). This is repeated in a hierarchical manner—i.e., collapsing the neighborhoods of nodes from the previous level—until the desired reduction ratio has been achieved.

b) Train a GCN on $\mathcal{G}_L$ We use the coarsest graph $\mathcal{G}_L$, and the corresponding aggregated features X_L and labels Y_L, to learn a message passing algorithm f_ϕ and a classification function g_θ, as described in Section 2.1. We note that any GCN can be used here.

c) Use the trained GCN to obtain embeddings at each level of the hierarchy Once training is complete, discard the classification function g_θ and use f_ϕ to learn embeddings $H_\ell \in \mathbb{R}^{|\mathcal{V}_\ell \times d|}$ at each level ℓ in the hierarchy. We use our sequence of projection matrices P_ℓ to lift these embeddings to $H'_\ell \in \mathbb{R}^{|\mathcal{V}| \times d}$ as $H'_\ell = P_1^+ \cdots P_{\ell+1}^+ H_\ell$, where $P_\ell^+[i,j] = 1$ iff $P_\ell[i,j] > 0$. This differs from Huang et al. (2021), who only learn embeddings for the original graph $\mathcal{G}$.

d) Aggregate the embeddings and learn a new classification function Each node $v_i \in \mathcal{V}$ is now associated with a sequence $h'_{0,i}, \ldots, h'_{L,i}$ of embeddings. We can aggregate these embeddings to obtain a final representation $\widetilde{h}_i = \textsc{combine}(h'_{0,i}, \ldots, h'_{L,i})$ for each node v_i, and learn a new classification function $g_{\tilde{\theta}}$ based on this new embedding. A number of choices can be made here. A lightweight option is to either concatenate the embeddings, or take a simple mean. Alternatively, if computational resources allow, we can learn a weighted average alongside the classification function, i.e. $\widetilde{h}_i = \sum_{\ell=0}^{L} w_\ell^T h'_{\ell,i}$. Unless otherwise stated, we use this weighted average approach; we explore alternatives in Appendix C.

Algorithm 1 SMGRL

input Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, features X, labels Y, hierarchical coarsening algorithm COARSE, embedding aggregation method COMBINE, GCN f_ϕ, classification function g_θ.

output Node embeddings $\widetilde{h}_i$, trained GCN f_ϕ, trained classification function $g_{\tilde{\theta}}$

1: $((\mathcal{G}_\ell, X_\ell, Y_\ell))_{\ell=0}^{L} = \text{COARSE}(\mathcal{G}, X, Y)$
2: Learn parameters ϕ and θ to minimize $\sum_{i=1}^{|\mathcal{V}_L|} \mathcal{L}\left(g_\theta\left(f_\phi(x_{L,i}, \{x_{L,j} : v_{L,j} \in \text{Ne}(v_i)\}\right)\right)$
3: **for** $\ell = 0, \ldots, L$ **do**
4: Obtain embeddings $h_{\ell,i}$ by applying f_ϕ to $\mathcal{G}_\ell$.
5: **end for**
6: Propogate embeddings such that $H'_\ell = P_1^+ \cdots P_{\ell+1}^+ H_\ell$
7: Learn parameters $\tilde{\theta}$ (and if appropriate, parameters of COMBINE$(\cdot)$) to minimize $\sum_{i=1,\ldots,|\mathcal{V}_0|} \mathcal{L}\left(g_{\tilde{\theta}}(\text{AGG}(h'_{0,i}, \ldots, h'_{L,i}), y_i\right)$
8: $\widetilde{H}_i \leftarrow \text{COMBINE}(H'_0, \ldots, H'_L)$

3.1 Discussion on the coarsening algorithm and applicability of SMGRL

We have chosen to use a method that retains spectral similarity, since spectrally coarsened graphs have been proven to be good proxies for training GCNs (Huang et al., 2021). However, we ideally *also* want our coarsened graphs to capture meaningful structure within the graph, allowing us to capture long-range dependencies. This sort of representation is what allows hierarchical GCNs such as Guo et al. (2021) and Jiang et al. (2020) to learn rich, multi-scale node embeddings.

To assess whether spectral graph coarsening yields an intuitive hierarchy, we applied spectral coarsening to a number of small, structured graphs: A stochastic blockmodel with 60 nodes (split into clusters of size 10, 20, 30) with edge probability matrix $0.03 + 0.77\mathbf{I}$ (Holland et al., 1983); a circular ladder graph with 50 nodes; a Newman-Watts-Strogatz graph with 100 nodes (Newman et al., 2001), initial clique size of 5, and edge addition probability of 0.1; Zachary's karate club graph (Zachary, 1977); the character co-occurrence graph from *Les Misérables* (Knuth, 1993; Hugo, 1863); and an Erdős-Rényi $G(n, p)$ graph with 100 nodes and edge probability 0.2 (Gilbert, 1959). All graphs were generated using `networkx` (Hagberg et al., 2008).

In Figure 2 we show the hierarchies obtained on each dataset. The left hand side of each subfigure shows the original graph (visualized using `networkx` with a spring layout). Moving towards the right of each subfigure, we see increasingly coarsened graphs. The bottom row shows the coarsened graph; the top row shows the mapping between nodes in the original graph, and nodes in the coarsened graph.

We see that the coarsening algorithm tends to group nodes that are close in the graph, and merge clusters into a single node. For example, in the stochastic block model graph, the coarsest graph contains one node for each latent community (Figure 2a). In the circular ladder graph, the graph is coarsened into a circle, with nodes representing contiguous regions of the original graph (Figure 2b). This suggests that the coarser graphs in the hierarchy do contain relevant information about the graph structure.

This figure also gives us a clue as to what sort of graphs might be well-suited to SMGRL. Figure 2a shows a stochastic blockmodel, which exhibits three clear clusters and no meaningful structure within clusters. We see that the coarsened graphs preserve the clustering structure, with $\mathcal{G}_2$ providing one node per original cluster. We see similar behavior in the Karate club social network (Figure 2d) and the Les Misérables co-occurrence graph (Figure 2e), which also show clear clustering structure. In the circular ladder graph (Figure 2b), we do not have clusters of nodes, but we do have a clear sense of distance between nodes, indicated by position along the ladder. Here, the coarsened graphs compress this distance, splitting the nodes based on position within the circle. We see similar behavior in the Newman-Watts-Strogatz graph (Figure 2c), which features several "chains" of nodes.

By contrast, the Erdős-Rényi graph (Figure 2f) exhibits neither clustering structure nor chains of nodes. The lack of structure means there is no "natural" clustering of the graph, and as a result the coarsened graphs do not capture meaningful information. In addition, some graphs cannot be meaningfully reduced via our

approach. In particular, a star graph (where one central node is connected to all other nodes, but no other edges are present) cannot be meaningfully condensed.

These observations suggest that SMGRL will work well on graphs which have some clustering structure, or where there is a significant variation in the distance between two nodes. It will not be appropriate in graphs with no or little structure, since we should not expect coarsening to capture any meaningful information about the graph.

4 Relationship to other hierarchical node embedding methods

SMGRL uses a hierarchical representation of a graph to learn multi-level embeddings, by first training a GCN on the coarsest level of the hierarchy, and then deploying that GCN at all levels. This approach falls under the general family of hierarchical node embeddings, as described in Section 2.2, and shares similarities with several other hierarchical or coarsening-based approaches. In this section, we discuss the most similar methods in detail, and highlight how SMGRL differs from these approaches.

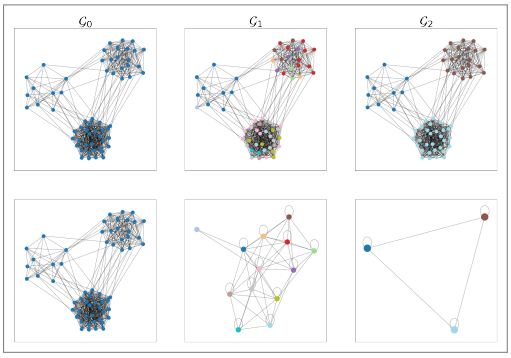
SMGRL makes use of the spectral coarsening algorithm of Loukas (2019). This algorithm was also used by Huang et al. (2021), who train a GCN on the coarsened graph and deploy it on the original graph. Unlike SMGRL, this approach does not incorporate hierarchical information: the coarsened graph is only used to reduce the computational complexity of training, and the node embeddings obtained on the coarsened graph are discarded. As we see in Section 5, this reduces the expressivity of the representations by ignoring longer-range or multi-scale relationships.

While Huang et al. (2021) only uses a hierarchical coarsening for computational reasons, other node embedding approaches do incorporate multi-level representations, either by explicitly aggregating representations learned at multiple resolutions or by sequentially refining representations across multiple levels of the hierarchy, as described in Section 2.2. A key advantage of SMGRL is the low computational cost of training: we train a single GCN on a single graph, and then deploy it at multiple levels. This is in contrast to methods such as GOSH (Akyildiz et al., 2020), MILE (Liang et al., 2021) or HARP (Chen et al., 2018), which iteratively refine embeddings as we descend the hierarchy, or methods that jointly learn separate GCNs across the entire hierarchy (Guo et al., 2021; Jiang et al., 2020).

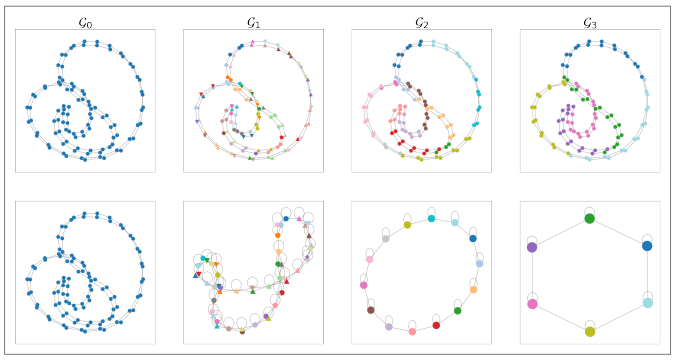
Of the alternative hierarchical node embeddings, MILE is perhaps most similar to SMGRL. Like SMGRL, it begins by learning a hierarchical coarsening, and learning embeddings on the coarsened graphs. However, these embeddings are learned using a transductive method such as node2vec, meaning the trained model cannot be reused at different levels of the hierarchy. Instead, they learn a GCN to refine the embeddings at each level of the hierarchy: nodes are initialized by projecting the previous level's final embeddings, and updated using a GCN. This GCN is shared among all levels, as in SMGRL; however, unlike our approach, MILE does not choose a coarsening designed to encourage the GCN to be generally applicable. Finally, MILE only uses the refined embeddings corresponding to the original graphs, discarding the intermediate embeddings; conversely, we combine embeddings from each level of resolution, making sure we retain information from coarser granularities.

5 Experiments

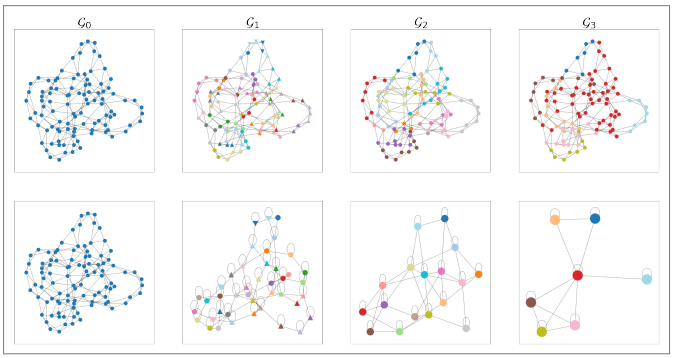
SMGLR is designed to improve upon existing GCN-based node representation-learning algorithms by learning and aggregating multi-resolution embeddings, without incurring excessive computational costs. In this section, we provide empirical evidence that our hierarchical representation increases expressivity (Sections 5.2 and 5.3). We show that training a single GCN and applying it at multiple resolution yields comparable performance to training layer-specific GCNs, while reducing computational cost and facilitating simple aggregation schemes. Indeed, we show that SMGLR outperforms several recent alternative hierarchical embedding methods (Section 5.5). We also show that SMGLR can be used in an inductive setting (Sec 5.6) and explore a distributed variant for settings where inference on the full graph is not feasible (Section 5.7).



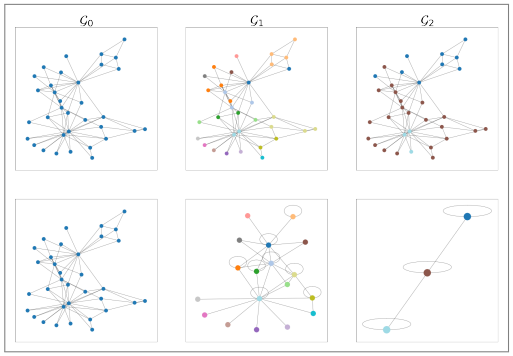
(a) Stochastic blockmodel with 60 nodes (split into clusters of size 10, 20, 30) with edge probability matrix $0.03 + 0.77\mathbf{I}$.



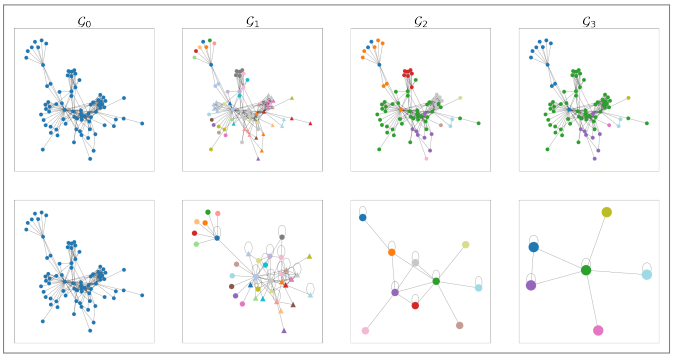
(b) Circular ladder graph with 50 nodes.



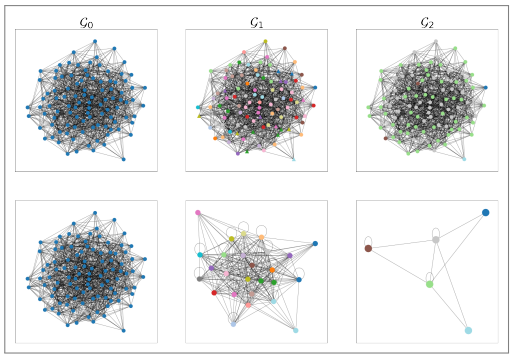
(c) Newman-Watts-Strogatz with 100 nodes, initial clique size of 5, and edge addition probability of 0.1.



(d) Karate club social network.



(e) *Les Misérables* co-occurence graph.



(f) Erdős-Rényi social network, with $n = 100, p = 0.2$.

Figure 2: Graph hierarchies obtained on various graphs, using a coarsening ratio of 0.95. Top row: Original graph $\mathcal{G}_0$, with node colors corresponding to the parent node in $\mathcal{G}_n$. Bottom row: Coarsened graph $\mathcal{G}_n$. Colors and shapes are consistent within columns of each subfigure: For example, the red circular nodes in the top row of column $\mathcal{G}_1$ correspond to the red circular node in the bottom row of column $\mathcal{G}_1$. Colors are not consistent between columns.

5.1 Datasets and implementation details

We consider seven multi-label classification datasets, summarized in Table 1. The first five datasets are those used by Huang et al. (2021), our closest comparison method. The last two datasets are much larger graphs from the Open Graph Benchmark dataset (Hu et al., 2020). To assess classification performance on these datasets, we look at the macro F_1 score on test-set labels; considering alternative metrics such as accuracy showed similar trends. Unless otherwise stated, we use the default train/validation/test split associated with each dataset for evaluating the macro F_1 score.

We apply SMGRL to three popular GCNs: GraphSAGE Hamilton et al. (2017), a GCN that uses skip connections to construct a generalized neighborhood aggregation scheme; APPNP (Klicpera et al., 2019), a GCN that uses an aggregation scheme inspired by personalized PageRank; and self-supervised graph attention networks (SuperGAT, Kim & Oh, 2021), a GCN that incorporates attention-based aggregation and uses self-supervised learning in training. These three methods are designed to show that SMGRL is applicable across a wide range of GCNs, incorporating different notions of neighborhood and different message aggregation schemes. We use existing Pytorch implementations of GraphSAGE, APPNP and SuperGAT[2]. To obtain class predictions from embeddings, we use a single-layer perceptron neural network.

We construct our hierarchy using the variation neighborhoods coarsening method (Loukas, 2019), using the original author's implementation[3]. This approach automatically determines the appropriate hierarchical depth, based on the desired reduction ratio. Unless otherwise stated, we learn per-dimension weights for each embedding to aggregate the multi-resolution embeddings. These weights are learned jointly with our final classification algorithm. In Appendix C, we explore some alternative aggregation schemes.

For both GraphSAGE and APPNP, we use additive aggregation of messages; in SuperGAT, an attention mechanism is used for message aggregation. For all three GCNs, we optimize using RMSProp with default hyperparameters. For APPNP, we use parameters $k = 3$ and $\alpha = 0.5$; for all other methods including HARP and MILE, we use the default parameters suggested by the authors. We carry out early stopping if validation loss does not improve in twenty epochs, and evaluate using the checkpoint with the best validation loss. For Cora, CiteSeer, PubMed, DBLP and Coauthor Physics, results are averaged over 200 runs with random seeds. For the larger OGBN-arXiv and OGBN-Products datasets, results are averaged over three runs with random seeds. Code to reproduce our experiments is available at `https://anonymous.4open.science/r/SMGRL`.

Table 1: Dataset overview

Name	Nodes	Edges	Classes	Features
Cora	2,708	10,556	7	1,433
CiteSeer	3,327	9,104	6	3,703
PubMed	19,717	88,648	3	500
DBLP	17,716	105,734	4	1,639
Coauthor Physics	34,493	495,924	5	8,415
ogbn-arxiv	169,343	1,166,243	40	128
ogbn-products	2,449,029	61,859,140	47	100

5.2 Incorporating representations at multiple resolutions leads to better embeddings

A single-layer GCN is only able to capture information about its immediate neighbors, which limits our ability to capture long-range graph signals. This is also true in the coarsened GCN approach of Huang et al. (2021); while a coarsened graph is used to *train* a GCN, the final embeddings are only obtained based on the full graph, so if we use a single-layer GCN we can only aggregate information from a direct neighborhood.

A key motivation behind SMGRL is that incorporating embeddings learned at multiple scales will lead to more informative representations and better performance on downstream tasks. To explore this, we begin by looking at the embeddings associated with each level in the hierarchy. Figure 3 shows test set macro F_1 score

[2] `https://pytorch-geometric.readthedocs.io`
[3] `https://github.com/loukasa/graph-coarsening`

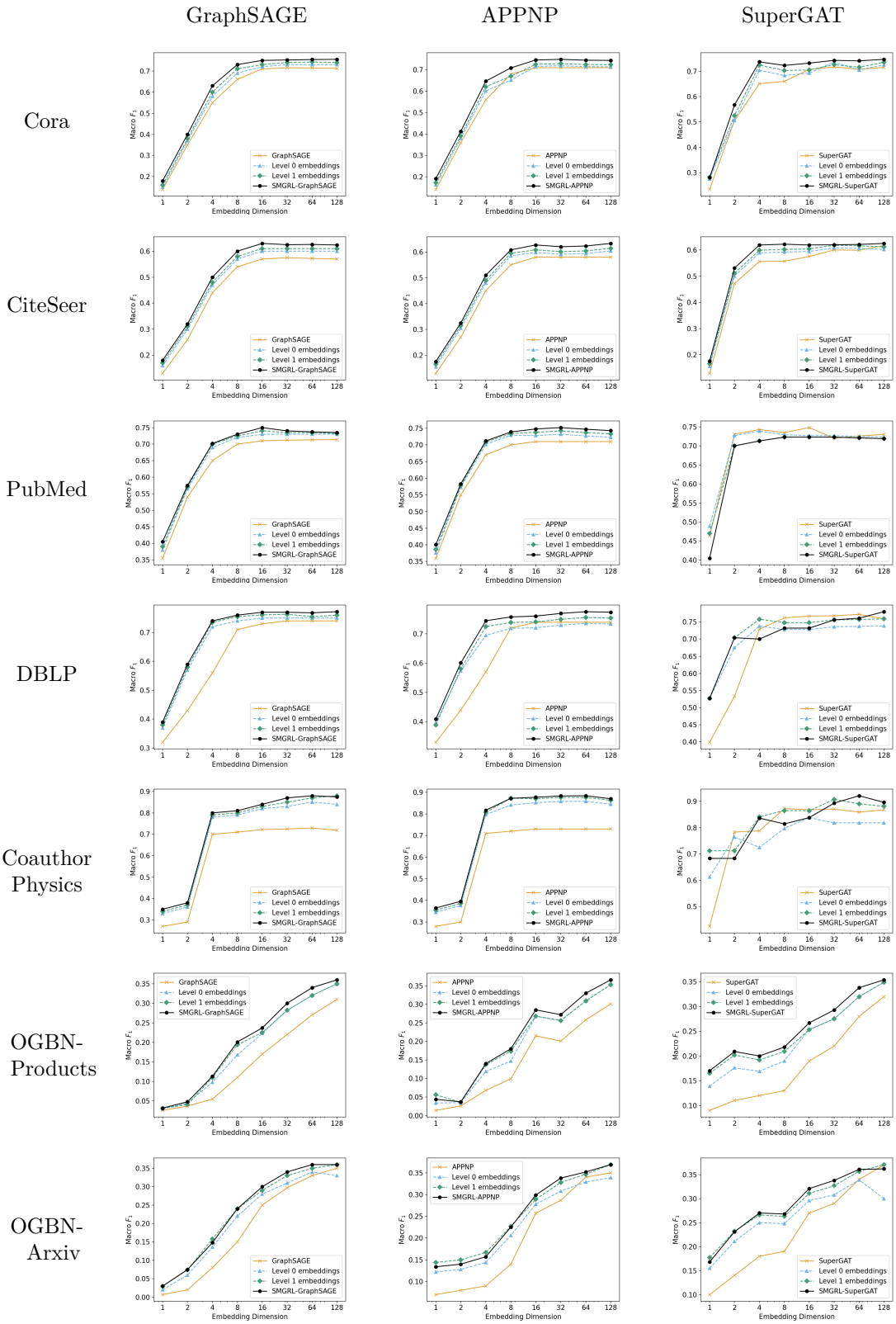


Figure 3: Test-set macro F_1 score for SMLRG embeddings obtained at different levels of the hierarchy, using three single-layer GCN architectures (GraphSAGE, APPNP, SuperGAT), a reduction ratio of 0.4 (resulting in a two-layer hierarchy), and various embedding dimensions.

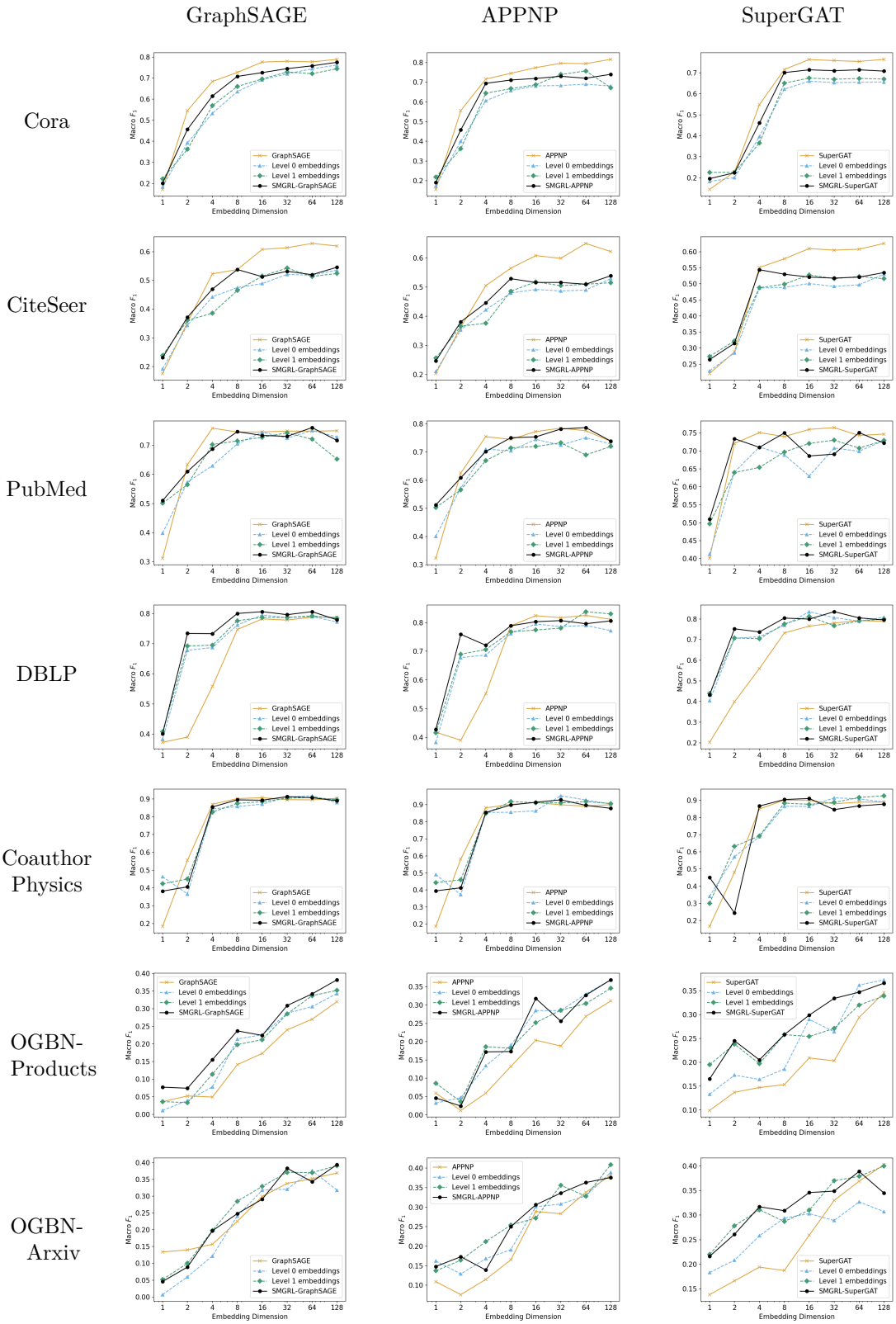


Figure 4: Test-set macro F_1 score for SMLRG embeddings obtained at different levels of the hierarchy, using three two-layer GCN architectures (GraphSAGE, APPNP, SuperGAT), a reduction ratio of 0.4 (resulting in a two-layer hierarchy), and various embedding dimensions.

across four datasets, using a reduction ratio of 0.4, and with a variety of embedding dimensions. Each column corresponds to one of the three base GCN algorithms, and each row to a dataset. Within each figure, the dashed lines correspond to embeddings associated with the coarsened graph (level 1) and the original graph (level 0). Recall that both of these embeddings are obtained using the base GCN trained on the coarsened (level 1) graph. The level 0 embedding is therefore equivalent to the graph coarsened approach of Huang et al. (2021). The solid black line corresponds to the SMGRL embedding obtained using a weighted average of the level 0 and level 1 embeddings. In addition, the solid orange line corresponds to embeddings obtained using the base GCN trained on the full graph. In Appendix B, we show comparable results holding embeddings dimension fixed and adjusting the reduction ratio (Figure 9).

We see several interesting patterns here. First, we note that learning either GraphSAGE or APPNP on the coarsened graph and then deploying it on the full graph (level 0) typically outperforms the corresponding baseline GCN. This supports the findings of Huang et al. (2021), who hypothesize that the coarsened graph acts as a regularizer, due to limiting the space of possible convolution operators. Performance is more mixed in the case of SuperGAT: while the level 0 embeddings outperform the full SuperGAT implementation for CiteSeer, OGBN-Products and OGBN-Arxiv, the full SuperGAT performs as well as or better than the level 0 embeddings on the other datasets. This may be because the attention-based aggregation scheme used in SuperGAT is quite distinct from the formulation in Kipf & Welling (2017) (which corresponds to approximate spectral convolution), suggesting that the algorithm might no longer be approximating spectral convolution. If this is the case, then there is less of a clear motivation to train on a spectrally coarsened graph.

Interestingly, in almost all cases where the level-0 embeddings outperform the full GCN, we find that using just the top-level embeddings—i.e., assigning to each node in $\mathcal{G}$ the embeddings obtained for its ancestor in the coarsened graph—actually performs *better* than the embeddings obtained using the same algorithm on the original graph. We hypothesize that this is because the lengthscale associated with the coarser graph may be better aligned with the natural variation in labels.

While the individual embeddings obtain good performance, we find that the full SMGRL embedding (that combines the embeddings from multiple layers using learned weights) tends to lead to better predictive performance than any single layer (although again, performance is more mixed for SuperGAT). This is not surprising: we are able to aggregate information across multiple lengthscales. In principle, if a given level of the hierarchy does not contain relevant information, we can learn to downweight embeddings from that level in the final aggregation (although a learned aggregation is not guaranteed to outperform any single layer on the test set; for example in the SuperGAT experiments we see several examples where a single layer outperforms the aggregation).

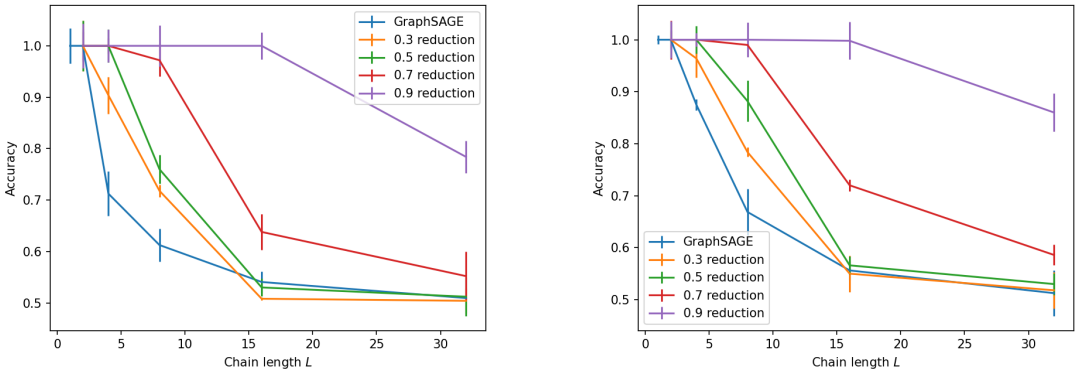
In Figure 4, we repeat this analysis using two-layer GraphSAGE, APPNP and SuperGAT. In general, we see better performance with an additional layer (albeit at a higher computational cost). In particular, the performance of GCNs trained on the full graph has improved over Figure 3, due to the ability to capture longer-range dependencies. As a result, SMGRL is no longer the clear winner in terms of performance, with the unmodified GCNs performing better in many (but not all) cases. However, the faster SMLRG method remains competitive with the full model, and as before, performs better than single-layer embeddings.

To summarize: SMGRL provides a lower cost alternative to full GCN training and inference. Where the full GCN model is able to capture long-range variation across the graph, SMGRL provides comparable performance. When the full GCN lacks this ability, SMGRL outperforms the full GCN.

5.3 SMGRL can capture long-range dependencies

Above, we hypothesized that, when used with a single-layer GCN, SMGRL embeddings are able to perform better than embeddings obtained on the original graph because they are able to capture variation at multiple degrees of resolution. Rather than just propagating information between neighboring nodes, they are also able to share information between related cliques and communities. This means each embedding is able to capture information over a larger total neighborhood. To demonstrate this, we consider a synthetic chain dataset proposed by Gu et al. (2020) to explore ability to capture long-range dependency. Each dataset contains 600 chain graphs of length L, with labels equally split across two classes. Within each chain, all nodes share the same label. All $100d$ features are non-informative, with the exception of the last node in

each chain, whose first two dimensions encode the true label. As we increase the chain length L, we increase the importance of long-range dependencies, by reducing the average distance to an informative node feature.



(a) One-layer GraphSAGE.

(b) Two-layer GraphSAGE.

Figure 5: Accuracy on a synthetic chain dataset, with various chain lengths and reduction ratios.

Figure 5a shows how accuracy varies with chain length L, applying SMGRL with a single-layer GraphSAGE architecture and varying reduction ratios.[4] For each dataset, we use 20 nodes for training, 100 nodes for validation, and 100 nodes for test. For a chain length of 2, single-layer GraphSAGE is able to achieve 100% accuracy, since all nodes include an informative node in their one-hop neighborhood. As we increase the chain length, the average performance of GraphSAGE decreases, as increasingly many nodes lack information in their one-hop neighborhood. However, SMGRL is able to capture longer-ranging dependencies. With a reduction ratio of 0.5, all length-4 chains are compressed to a length-2 chain in the coarsest graph, and in this representation all nodes contain relevant information in their one-hop neighborhood. As we increase the reduction ratio further, the top-level nodes are increasingly likely to include relevant information in their one-hop neighborhood, leading to increased accuracy. We see a similar pattern in Figure 5b, where the underlying architecture is a two-layer GraphSAGE. Here, the baseline algorithm can propagate information from the two-hop neighborhood, and therefore obtains better accuracies than the one-layer counterpart. The general pattern seen in the one-layer case is repeated, however: as the average relevant lengthscale increases beyond 2, we see improved performance by incorporating longer-range dependencies using SMGRL.

5.4 Learning a single GCN does not sacrifice embedding quality

Clearly, incorporating information obtained at multiple levels of granularity can improve performance over a single level of granularity—whether that level is the original graph, or a single coarsened layer. However, under SMGRL, only one level in the hierarchy has embeddings obtained using a GCN trained on that level. A reasonable question might be whether training one GCN per layer, and combining the resulting embedding, does better than our approach. In Table 2 we do exactly that: on the Cora dataset, we compare the SMGRL framework with a single GCN learned at the top level, with a variant that learns a separate GCN for each level. When using our standard weighting aggregation scheme, or using concatenated embeddings, we perform comparably to this variant, for a significantly lower training cost. When using a simple mean aggregation, we uniformly outperform the separate GCNs. This is because using a single GCN means that the embeddings at the two levels are inherently compatible—they aggregate information in the same manner, so each dimension carries similar information. Conversely, two independently trained GCNs will not lead to compatible embeddings, leading to greater loss of information when taking a simple mean.

Table 2: Macro F_1 score on Cora datase, with a reduction ratio of 0.4 and various embedding dimensions and aggregation methods. Top: using a single one-layer GraphSAGE model learned on the coarsest level. Bottom: using a separate one-layer GraphSAGE model learned at each level. Stated dimensions correspond to per-level embeddings; concatenated embeddings are twice this. We consider three aggregation methods, plus the embeddings obtained on the original graph (equivalent to the coarsened GCN model of Huang et al. (2021)). For each embedding type, we bold the better of the two results across the two models. Note that the stated embedding dimension is that of the per-level embeddings; the concatenated embeddings are twice this length.

		Embedding dimension							
	Aggregation method	1	2	4	8	16	32	64	128
Single GCN trained on coarsest level	Simple mean	**0.184**	**0.392**	**0.621**	**0.731**	**0.755**	**0.761**	**0.765**	**0.761**
	Weighted mean	0.181	0.404	**0.633**	**0.736**	0.757	0.762	**0.769**	**0.768**
	Concatenation	0.22	0.398	0.615	0.728	**0.755**	**0.759**	**0.757**	**0.749**
Separate GCN per level	Simple mean	0.18	0.372	0.574	0.695	0.741	0.754	0.756	0.747
	Weighted mean	**0.201**	**0.419**	0.628	0.73	**0.758**	**0.763**	0.765	0.765
	Concatenation	**0.352**	**0.569**	**0.692**	**0.742**	0.754	0.755	0.743	0.741

5.5 SMGRL outperforms alternative hierarchical methods

In Figure 6, we compare against two popular hierarchical graph embedding methods, HARP (Chen et al., 2018) and MILE (Liang et al., 2021), implemented using the authors' original code[5][6] with default settings and the same classification architecture as SMGRL. HARP generates embeddings on a coarsened graph using node2vec, and refines them as we descent the hierarchy. MILE generates embeddings on a coarsened graph using NetMF (Qiu et al., 2018), and uses a (globally shared) GCN to refine. Since these approaches incorporate both an initial embedding and a refinement, they have higher runtimes than SMGRL. Despite this, we see that SMGRL outperforms both approaches for most settings.

5.6 SMGRL can be used in an inductive manner

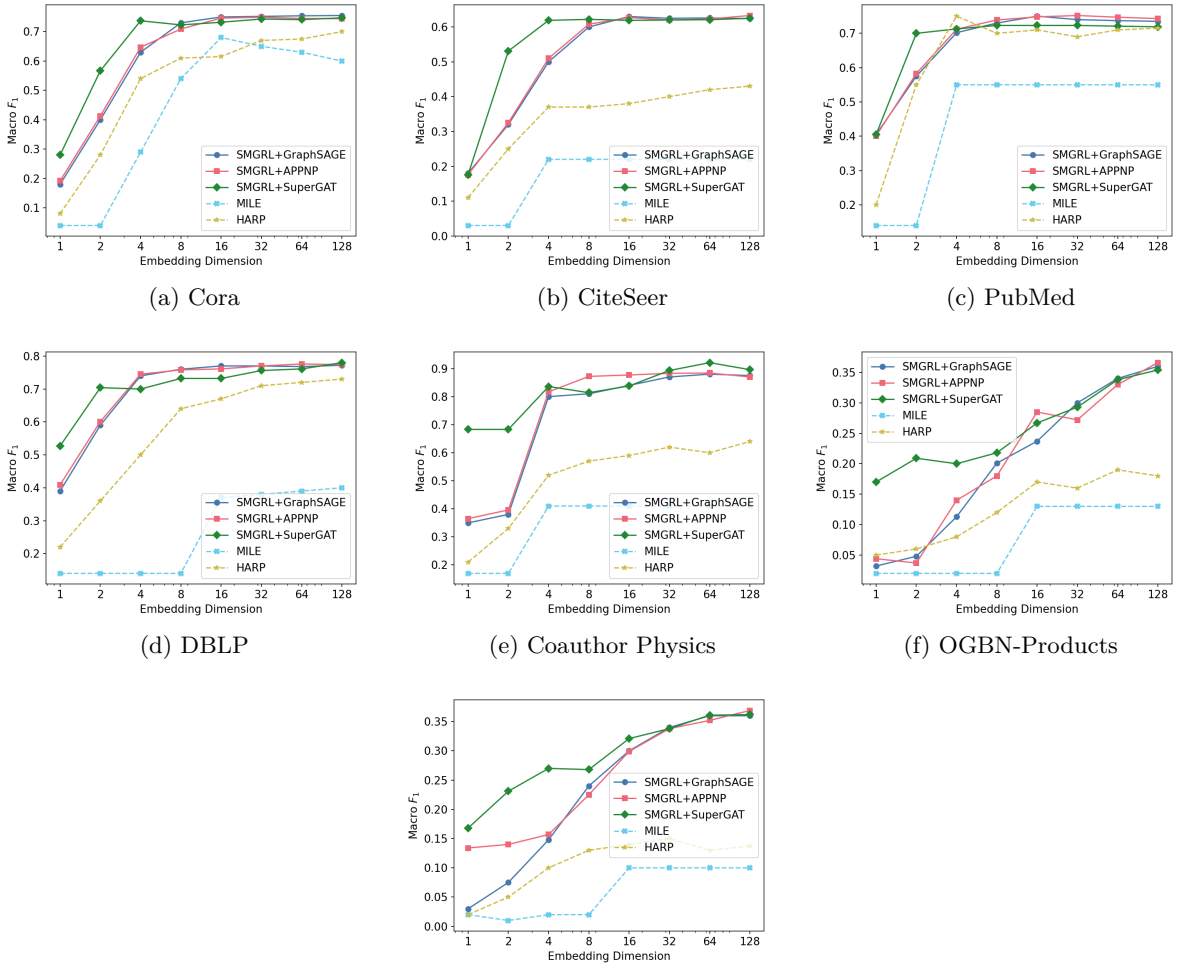
A key advantage of GCNs over other node embedding methods is that they are inductive. Rather than learn a direct mapping from node to embedding, GCNs learn message passing algorithms that can be applied to arbitrary graphs. Indeed, we directly make use of this when we apply an algorithm trained at one level of the hierarchy, to get embeddings at other levels.

We consider a mixed inductive/transductive setting, where a randomly selected subset of test set nodes are not included in the training graph. We refer to the held-out test set nodes as our inductive test set, and the test set nodes present at training time as our transductive test set. At inference time, we reincorporate the held-out nodes at each level in our hierarchy. At the bottom level, we simply reintroduce the nodes and any missing edges. At higher levels, we assign the new nodes to the partitions with which they have the highest number of edges. In Figure 7, we show macro F_1 on Cora scores for a reduction ratio of 0.4, 8-dimensional embeddings, and a variety of held-out percentages. Error bars show one standard deviation under different random held-out sets. While we do perform better on the transductive test set than the inductive test set, the difference is small relative to the variation across partitions, indicating that SMGRL continues to perform well when a moderate number of new nodes are added to the graph.

5.7 Distributing inference for large graphs

SMGRL dramatically reduces training time of a GCN, since training is carried out over a smaller graph. However, inferring the final embeddings occurs on the full graph (plus any additional, smaller layers in the

[4]GraphSAGE on the full graph corresponds to a reduction ratio of 0.
[5]`https://github.com/jiongqian/MILE`
[6]`https://github.com/GTmac/HARP`



(a) Cora (b) CiteSeer (c) PubMed

(d) DBLP (e) Coauthor Physics (f) OGBN-Products

(g) OGBN-Arxiv

Figure 6: Macro F_1 score on four different graphs, with varying embedding dimensions, for various hierarchical embedding methods. SMGRL uses GraphSAGE with a single-layer architecture, with a reduction ratio of 0.4.

hierarchy). While, unlike training, inference only requires a single pass through the graph, this inference cost—and the associated memory requirements—may be infeasible if our graph is large.

In such settings, we propose partitioning the graph at each level of the hierarchy, based on their parents in the graph, to approximate the full graph with a sequence of independent subgraphs. We can then infer the embeddings on each subgraph in parallel, making use of distributed resources if available. This leads to lower computational complexity and memory requirements, but will tend to reduce the quality of the embeddings, since we are ignoring many edges in the original graph. In Figure 8, we compare the macro F_1 score of embeddings obtained using the disjoint subgraphs at lower levels, and the embeddings obtained using the full graph, on the Cora dataset. We see that, while there is a drop in performance due to using the disjoint subsets, it is small and may be a worthwhile trade-off in practice.

6 Discussion

In this work we introduce SMGRL, an extremely lightweight framework for learning multi-resolution node embeddings. SMGRL is highly customizable and can be applied on top of any GCN. Moreover, its computational cost is approximately equivalent to that of the recently proposed, coarsening-based scalable GCN framework

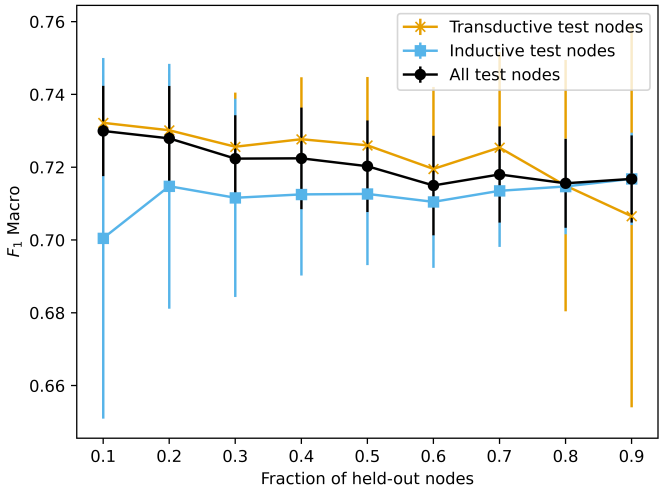


Figure 7: Inductive learning on Cora using SMLRG-GraphSAGE, with a varying percentage of test set nodes held out during training. Macro F_1 plotted for overall test set, and for the inductive and transductive subsets.

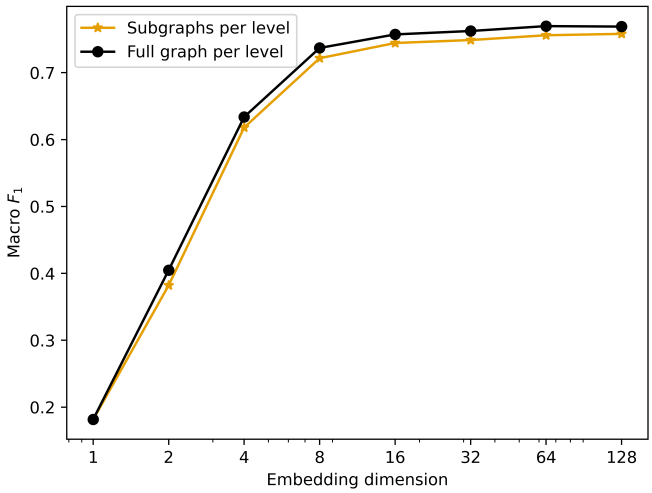


Figure 8: Cora macro F_1 score, using full SMGRL-GraphSAGE, and a variant where inference is carried out on a partitioned graph.

of Huang et al. (2021), while introducing increased expressive power due to incorporating information at multiple lengthscales. Our experiments on real-world datasets show impressive empirical performance. A future direction might be to incorporate pre-existing partitioning information, such as partitioning users of a social network by employer or country.

References

Taha Atahan Akyildiz, Amro Alabsi Aljundi, and Kamer Kaya. GOSH: Embedding big graphs on small hardware. In *International Conference on Parallel Processing-ICPP*, 2020.

Sambaran Bandyopadhyay, Manasvi Aggarwal, and M Narasimha Murty. Self-supervised hierarchical graph neural network for graph representation. In *IEEE International Conference on Big Data*, 2020.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional ARMA filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Derrick Blakely, Jack Lanchantin, and Yanjun Qi. Time and space complexity of graph convolutional networks. Accessed on: July 11, 2023.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

Gecia Bravo Hermsdorff and Lee Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. In *Advances in Neural Information Processing Systems*, 2019.

Marc Brockschmidt. GNN-film: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning*, 2020.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *International Conference on Learning Representations*, 2021.

Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: Hierarchical representation learning for networks. In *AAAI Conference on Artificial Intelligence*, 2018.

Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Knowledge Discovery and Data Mining*, 2016.

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In *Advances in Neural Information Processing Systems*, 2020.

Kan Guo, Yongli Hu, Yanfeng Sun, Sean Qian, Junbin Gao, and Baocai Yin. Hierarchical graph convolution networks for traffic forecasting. In *AAAI Conference on Artificial Intelligence*, 2021.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, 2008.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Fenyu Hu, Yanqiao Zhu, Shu Wu, Liang Wang, and Tieniu Tan. Hierarchical graph convolutional networks for semi-supervised node classification. In *International Joint Conference on Artificial Intelligence*, 2019.

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 *Advances in Neural Information Processing Systems*, 2020.

Jingjia Huang, Zhangheng Li, Nannan Li, Shan Liu, and Ge Li. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *International Conference on Computer Vision*, 2019.

Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Knowledge Discovery and Data Mining*, 2021.

Victor Hugo. *Les misérables*. C. Lassalle, 1863.

Hao Jiang, Peng Cao, , MingYi Xu, Jinzhu Yang, and Osmar Zaiane. Hi-GCN: A hierarchical graph convolution network for graph embedding learning of brain network and brain disorders prediction. *Computers in Biology and Medicine*, 127:104096, 2020.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021.

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

J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *International Conference on Learning Representations*, 2019.

Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*, volume 1. AcM Press New York, 1993.

Kangjie Li, Yixiong Feng, Yicong Gao, and Jian Qiu. Hierarchical graph attention networks for semi-supervised node classification. *Applied Intelligence*, 50(10):3441–3451, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. MILE: A multi-level framework for scalable graph embedding. In *International AAAI Conference on Web and Social Media*, 2021.

N. Liu, S. Jian, D. Li, Y. Zhang, Z. Lai, and H. Xu. Hierarchical adaptive pooling by capturing high-order dependency for graph representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 35 (4):3952–3965, 2021.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning*, 2018.

Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(2):026118, 2001.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2019.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Knowledge Discovery and Data Mining*, 2014.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, LINE, PTE, and node2vec. In *International Conference on Web Search and Data Mining*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Zhenghua Xin, Guolong Chen, Jie Chen, Shu Zhao, Zongchao Wang, Aidong Fang, Zhenggao Pan, and Lin Cui. Mgpool: multi-granular graph pooling convolutional networks representation learning. *International Journal of Machine Learning and Cybernetics*, 13:783–796, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, 2018.

Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.

Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. Hierarchical message-passing graph neural networks. *Data Mining and Knowledge Discovery*, 37:381–408, 2022.

A Computational complexity of SMGRL

We discuss the cost of each step of SMGRL below, noting that the first two steps are identical to the coarsened GCN framework of Huang et al. (2021)

- Constructing a coarsened graph using the neighborhood-based coarsening algorithm of Loukas (2019) scales (up to polylog terms) linearly with both the number of edges, and the product of the number of nodes and the average degree – both of which are upper bounded by $|\mathcal{V}|^2$.
- Training a GCN on the coarsest graph has computational complexity $O(K|\mathcal{V}_L| + K|\mathcal{E}_L|)$ (Blakely et al.), where K is the number of layers in the GCN. Using a coarsening method with reduction ratio r, the coarsened graph $\mathcal{G}_L$ has $|\mathcal{V}_L| = (1-r)|\mathcal{V}|$ vertices. The number of edges $|\mathcal{E}_L|$ in $\mathcal{G}_L$ is upper bounded by $\min\left(|\mathcal{E}|, (1-r)^2|\mathcal{V}|^2\right)$. If $\mathcal{G}$ is dense, then $|\mathcal{E}| \sim O(|\mathcal{V}|^2)$, suggesting a $(1-r)^2$ speed-up over the original GCN. If $\mathcal{G}$ is sparse then $|\mathcal{E}| \sim O(|\mathcal{V}|)$, suggesting a $(1-r)$ speed-up over the original GCN.
- Since the coarsened graphs are smaller than the original graph, for both SMGRL and the coarsened algorithm the inference step is dominated by the cost of inference on the original graph, which scales as $O\left(K|\mathcal{V}| + K|\mathcal{E}|\right)$. Since our hierarchy allows us to extract information at multiple resolutions by design, we choose to let the number of layers, K, to be one. In practice the number of iterations required for convergence of the GCN means that in the graphs considered in this paper, the cost of inference is smaller than the cost of training. If desired, inference can be approximated by partitioning the graphs at levels $\ell < L$ based on their parents in level $\ell + 1$, and independently inferring embeddings on the resulting subgraph. This has the additional advantage of allowing parallelization across subgraphs. We empirically explore the resulting performance in Section 5.7.
- Aggregating the embeddings and learning the final classification algorithm is linear in the number of labeled vertices $|\mathcal{V}|$, which is typically smaller than $|\mathcal{E}|$.

B Additional experiments comparing SMGRL with single-level embeddings

In Section 5.2, we compared how macro F_1 scores between a GCN on a single graph, SMGRL in conjunction with that GCN, and the per-level embeddings obtained using SMGRL in conjunction with that GCN, for a fixed reduction ratio and varying embedding dimensions. In Figure 9, we show equivalent plots holding the embedding dimension fixed at 8 and with varying reduction ratios. Note that the number of levels of the hierarchy are determined by the reduction ratio. As before, the level 0 embeddings are equivalent to the approach of Huang et al. (2021). We see a similar pattern: For all but the highest reduction ratio, the GCN learned on the coarsest graph performs well at all layers, typically leading to embeddings that perform better than those obtained using a GCN trained on the original graph (indicated by a reduction ratio of zero). And in almost all cases, an aggregated embedding which considers all levels of the hierarchy outperforms any single level's embeddings.

C Exploration of alternative final aggregation schemes

As we have seen in the experiments so far, learning post-hoc weights to combine the embeddings works well in practice. However, this does add additional computational complexity (albeit minimally, as described in Appendix A), and limits our choice of classifier to one that can be jointly trained with the post-hoc weightings. In Figure 10, we compare our post-hoc weighting scheme with two alternatives: taking a simple mean, and directly concatenating the embedding. All experiments use single-layer GraphSAGE, with a reducion ratio of 0.4 and varying embedding dimensions. Note that embedding dimension here refers to the per-layer embedding dimension; the concatenated dimension will be multiplied by the number of dimensions in the hierarchy. We see that in general, the learned weighted mean performs better than the alternative approaches; however the difference is fairly small. This leads us to suggest using a simple mean if computational resources are limited.

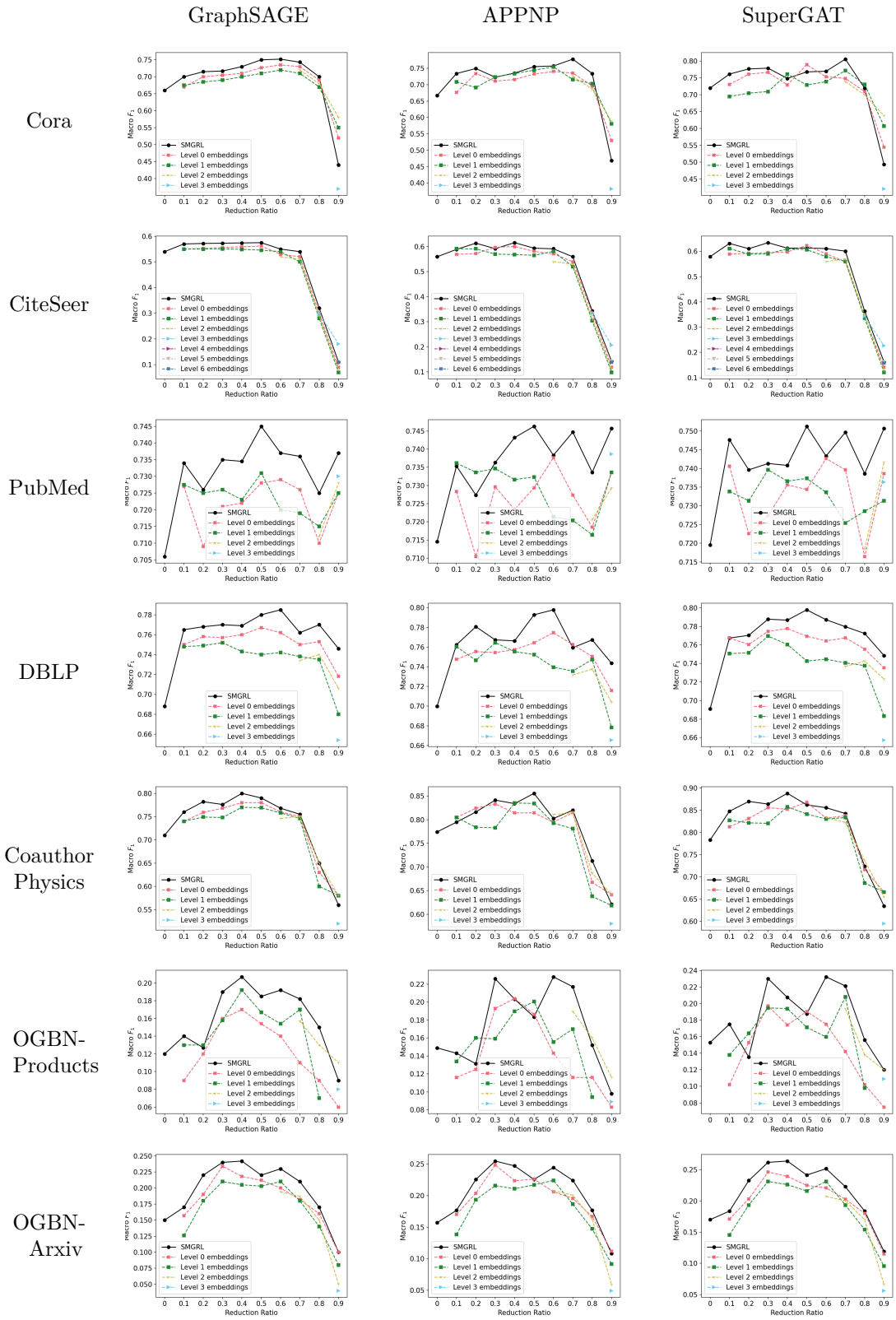
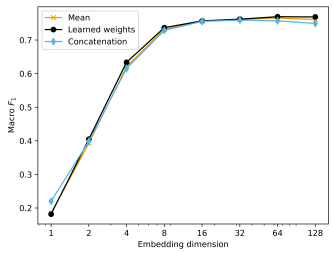
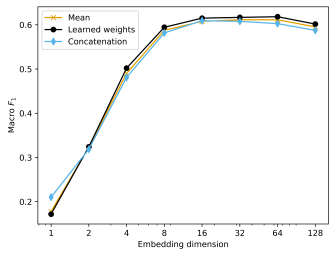


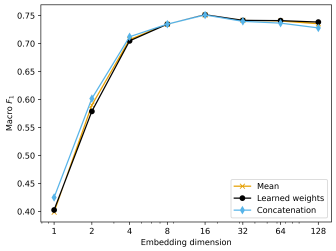
Figure 9: Test-set macro F_1 score for SMLRG embeddings obtained at different levels of the hierarchy, using three single-layer GCN architectures (GraphSAGE, APPNP, SuperGAT), an embedding dimension of 8, and various reduction ratios. Note that a reduction ratio of zero corresponds to the GCN on the original graph.



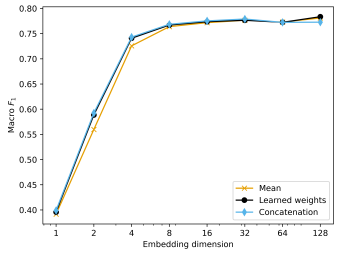
(a) Cora



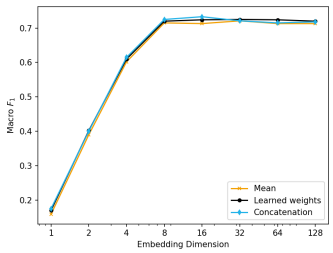
(b) CiteSeer



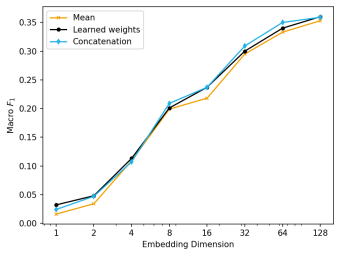
(c) PubMed



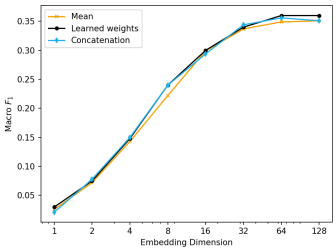
(d) DBLP



(e) Coauthor Physics



(f) OGBN-Products



(g) OGBN-Arxiv

Figure 10: Macro F_1 score on seven different graphs, for three different aggregation schemes, using SMGRL + single-layer GraphSAGE. The reduction ratio is fixed at 0.4, and a variety of embedding dimensions are shown.