Unsupervised Learning of Graph Hierarchical Abstractions with Differentiable Coarsening and Optimal Transport

Tengfei Ma* **Jie Chen***
MIT-IBM Watson AI Lab, IBM Research
Tengfei.Ma1@ibm.com chenjie@us.ibm.com

Abstract

Hierarchical abstractions are a methodology for solving large-scale graph problems in various disciplines. Coarsening is one such approach: it generates a pyramid of graphs whereby the one in the next level is a structural summary of the prior one. With a long history in scientific computing, many coarsening strategies were developed based on mathematically driven heuristics. Recently, resurgent interests exist in deep learning to design hierarchical methods learnable through differentiable parameterization. These approaches are paired with downstream tasks for supervised learning. In practice, however, supervised signals (e.g., labels) are scarce and are often laborious to obtain. In this work, we propose an unsupervised approach, coined OTCOARSENING, with the use of optimal transport. Both the coarsening matrix and the transport cost matrix are parameterized, so that an optimal coarsening strategy can be learned and tailored for a given set of graphs without use of labels. We demonstrate that the proposed approach produces meaningful coarse graphs and yields competitive performance compared with supervised methods for graph classification and regression.

Introduction

A proliferation of graph neural networks (Bruna et al. 2014; Henaff, Bruna, and LeCun 2015; Duvenaud et al. 2015; Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Chen, Ma, and Xiao 2018; Veličković et al. 2018; Ying et al. 2018a; Liao et al. 2019; Xu et al. 2019b; Scarselli et al. 2009; Li et al. 2016; Gilmer et al. 2017; Jin et al. 2017) emerged recently with wide spread applications ranging from theorem proving (Wang et al. 2017), chemoinformatics (Jin et al. 2017; Fout et al. 2017; Schütt et al. 2017), to planning (Ma et al. 2020). These models learn sophisticated feature representations of a graph and its constituents (i.e., nodes and edges) through layers of feature transformation. Several architectures (Xu et al. 2019b; Morris et al. 2019; Maron et al. 2019) are connected to the Weisfeiler–Lehman (WL) graph isomorphism test (Shervashidze et al. 2011) because of the resemblance in iterative node (re)labeling.

An image analog of graph neural networks is convolutional neural networks, whose key components are convolution and pooling. The pooling operation reduces the spatial dimensions of an image and forms a hierarchical abstraction through successive downsampling. For graphs, a similar hierarchical abstraction is particularly important for maintaining the structural information and deriving a faithful feature representation. A challenge, however, is that unlike image pixels that are spatially regular, graph nodes are irregularly connected and hence pooling is less straightforward.

Several graph neural networks perform pooling in a hierarchical manner. The work of Bruna et al. (2014) builds a multiresolution hierarchy of the graph with agglomerative clustering, based on ϵ-covering. The work of Defferrard, Bresson, and Vandergheynst (2016) and Fey et al. (2018) employ Graclus that successively coarsens a graph based on the heavy-edge matching heuristic. The work of Simonovsky and Komodakis (2017) constructs the hierarchy through a combined use of spectral polarity and Kron reduction. These neural networks build the graph hierarchy as preprocessing, which defines in advance how pooling is performed given a graph. No learnable parameters are attached.

Recently, hierarchical abstractions as a learnable neural network module surfaced in graph representation learning. Representative approaches include DIFFPOOL (Ying et al. 2018b), GRAPH U-NET (Gao and Ji 2019), and SAGPOOL (Lee, Lee, and Kang 2019). All approaches treat the learnable hierarchy as part of the neural network (in conjunction with a predictive model), which is trained with a downstream task in a (semi-)supervised manner.

In practice, however, supervised signals (e.g., labels) are scarce and are often laborious and expensive to obtain. Hence, in this work, we propose an unsupervised approach, called OTCOARSENING, that produces a hierarchical abstraction of a graph independent of downstream tasks. Therein, node features for the graphs in the hierarchy are derived simultaneously, so that they can be used for different tasks through training separate downstream predictive models. OTCOARSENING consists of two ingredients: a parameterized graph coarsening strategy in the algebraic multigrid (AMG) style; and an optimal transport that minimizes the structural transportation between two consecutive graphs in the hierarchy, thus replacing the cross-entropy or other losses that rely on labeling information. The "OT" part of the name comes from Optimal Transport. We show that this unsupervised approach produces meaningful coarse graphs

*These two authors contribute equally.

that are structure preserving; and that the learned representations perform competitively with supervised approaches.

The contribution of this work is threefold. First, for unsupervised learning we introduce a new technique based on hierarchical abstraction through minimizing discrepancy along the hierarchy. Second, key to a successful hierarchical abstraction is the coarsening strategy. We develop one motivated by AMG and empirically show that the resulting coarse graphs qualitatively preserve the graph structure. Third, we demonstrate that the proposed technique, combining coarsening and unsupervised learning, performs comparably with supervised approaches but is advantageous in practice facing label scarcity.

Related Work

Hierarchical (a.k.a. multilevel or multiscale) methods are behind the solutions of a variety of problems, particularly for graphs. Therein, coarsening approaches are being constantly developed and applied. Two active areas are graph partitioning and clustering. The former is often used in parallel processing, circuit design, and solutions of linear systems. The latter appears in descriptive data analysis.

Many of the graph hierarchical approaches consist of a coarsening and an uncoarsening phase. The coarsening phase successively reduces the size of a given graph, so that an easy solution can be obtained for the smallest one. Then, the small solution is lifted back to the original graph through successive refinement in the reverse coarsening order. For coarsening, a class of approaches applies heave-edge matching heuristics (Hendrickson and Leland 1995; Karypis and Kumar 1998; Dhillon, Guan, and Kulis 2007). Loukas and coauthors show that for certain graphs, the principal eigenvalues and eigenspaces of the coarsened and the original graph Laplacians are close under randomized matching (Loukas and Vandergheynst 2018; Loukas 2019). Bravo-Hermsdorff and Gunderson (2019) show that contracting two nodes into one may be interpreted as perturbing the Laplacian pseudoinverse with an infinitely weighted edge. On the other hand, in the uncoarsening phase, refinement can be done in several ways, including Kernighan-Lin refinement (Kernighan and Lin 1970; Shi and Malik 2000; Luxburg 2007) and kernel k-means (Dhillon, Guan, and Kulis 2007).

Another class of coarsening approaches selects a subset of nodes from the original graph. Call them coarse nodes; they form the node set of the coarse graph. Other nodes are aggregated with weights to the coarse nodes in certain ways, which, simultaneously define the edges in the coarse graph. Many of these methods were developed akin to algebraic multigrid (AMG) (Ruge and Stüben 1987), wherein the coarse nodes, the aggregation rule, and edge weights may be defined based on original edge weights (Kushnir, Galun, and Brandt 2006), diffusion distances (Livne and Brandt 2012), or algebraic distances (Ron, Safro, and Brandt 2011; Chen and Safro 2011; Safro, Sanders, and Schulz 2014). In this work, the selection of the coarse nodes and the aggregation weights are parameterized and learned instead.

Hierarchical graph representation is emerging in deep learning. Representative approaches include DIFFPOOL (Ying et al. 2018b), GRAPH U-NET (Gao and Ji 2019), and SAGPOOL (Lee, Lee, and Kang 2019). Cast in the above setting, DIFFPOOL is similar to the first class of coarsening approaches, whereas GRAPH U-NET and SAGPOOL similar to the latter. All methods are supervised, as opposed to ours.

Our work is additionally drawn upon optimal transport, a tool recently used for defining similarity of graphs (Vayer et al. 2019; Xu et al. 2019a). In the referenced work, Gromov–Wasserstein distances are developed that incorporate both node features and graph structures. Moreover, a transportation distance from the graph to its subgraph is developed by Garg and Jaakkola (2019). Our approach is based on a relatively simpler Wasserstein distance, whose calculation admits an iterative procedure more friendly to neural network parameterization.

Method

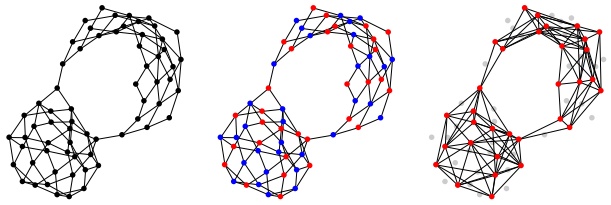
In this section, we present the proposed method OTCOARSENING, beginning with two main ingredients: coarsening and optimal transport, followed by a summary of the computational steps in training and the use of the results for downstream tasks.

AMG-Style Coarsening

The first ingredient coarsens a graph G into a smaller one G_c. For a differentiable parameterization, an operator will need be defined that transforms the corresponding graph adjacency matrix $A \in \mathbb{R}^{n\times n}$ into $A_c \in \mathbb{R}^{m\times m}$, where n and m are the number of nodes of G and G_c respectively, with $m < n$. We motivate the definition by algebraic multigrid (Ruge and Stüben 1987), because of the hierarchical connection and a graph-theoretic interpretation. AMG also happened to be referenced as a potential candidate for pooling in some graph neural network architectures (Bruna et al. 2014; Defferrard, Bresson, and Vandergheynst 2016).

Background on Algebraic Multigrid AMG belongs to the family of multigrid methods (Briggs, Henson, and McCormick 2000) for solving large, sparse linear systems of the form $Ax = b$, where A is the given sparse matrix, b is the right-hand vector, and x is the unknown vector to be solved for. For simplicity, we assume throughout that A is symmetric. The simplest algorithm, two-grid V-cycle, consists of the following steps: (i) Approximately solve the system with an inexpensive iterative method and obtain an approximate solution x'. Let $r = b - Ax'$ be the residual vector. (ii) Find a tall matrix $S \in \mathbb{R}^{n\times m}$ and solve the smaller residual system $(S^TAS)y = S^Tr$ for the shorter unknown vector y. (iii) Now we have a better approximate solution $x'' = x' + Sy$ to the original system. Repeat the above steps until the residual is sufficiently small.

The matrix of the residual system, S^TAS, is called the Galerkin coarse-grid operator. One may show that step (ii), if solved exactly, minimizes the energy norm of the error $x - x''$ over all possible corrections from the range of the matrix S. Decades of efforts on AMG discover practical definitions of S that both is economic to construct/apply and encourages fast convergence. We depart from these efforts



(a) Graph G (b) Coarse nodes (red) and fine nodes (blue) (c) Coarse graph G_c (using only half of the nodes in G)

Figure 1: Example graph and coarsening.

and define/parameterize an S that best suites graph representation learning.

Coarsening Framework Following the above motivation, we settle with the coarsening framework

$$A_c = S^T AS, \tag{1}$$

where S is named the *coarsening matrix*. For parameterization, we might have treated S as a parameter matrix, but it requires a fixed size to be learnable and hence it can only be applied to graphs of the same size. This restriction both is unnatural in practice and destroys permutation invariance of the nodes. In what follows, we discuss the properties of S from a graph theoretic view, which leads to a natural parameterization.

Properties of S Let V be the node set of the graph G. AMG partitions V into two disjoint subsets C and F, whose elements are called *coarse nodes* and *fine nodes*, respectively. See Figure 1(b). For coarsening, C becomes the node set of the coarse graph and the nodes in F are eliminated.

The rows of the coarsening matrix S correspond to the nodes in V and columns to nodes in C. This notion is consistent with definition (1), because the rows and columns of A_c correspond to the coarse nodes. It also distinguishes from DIFFPOOL (Ying et al. 2018b), which although defines the next graph by the same equation (1), does not use the nodes in the original graph as those of the smaller graph.

If S is dense, so is A_c. Then, the graphs in the coarsening hierarchy are all complete graphs, which are less desirable. Hence, we would like S to be sparse. Assuming so, one sees that each column of S plays the role of aggregation. For convenience, we define $\chi(j)$ to be the set of nonzero locations of this column and call it the *aggregation set* of the coarse node j. The following result characterizes the existence of an edge in the coarse graph.

Theorem 1. *There is an edge connecting two nodes j and j' in the coarse graph if and only if there is an edge connecting the two aggregation sets $\chi(j)$ and $\chi(j')$ in the original graph.*

Proof. We say that the sum of two numbers is *structurally nonzero* if at least one of the numbers is nonzero, even if they sum algebraically to zero (e.g., when one number is the opposite number of the other). Structural nonzero of an element in the adjacency matrix is the necessary and sufficient condition for the existence of the corresponding edge in the graph.

Recall that $A_c = S^T AS$. For two coarse nodes j and j', one sees that the element $A_c(j, j')$ is structurally nonzero if and only if the submatrix $A(\chi(j), \chi(j'))$ is nonempty. In other words, j and j' are connected by an edge in the coarse graph G_c if and only if there exists an edge connecting $\chi(j)$ and $\chi(j')$ in the original graph G. Note that such an edge may be a self loop. □

Hence, in order to encourage sparsity of the coarse graph, many of the aggregation set pairs should not be connected by an edge. One principled approach to ensuring so, is to restrict the aggregation set to contain at most direct neighbors and the node itself. The following corollary is straightforward. We say that the *distance* of two nodes is the number of edges in the shortest path connecting them.

Corollary 2. *If each aggregation set contains at most direct neighbors and the node itself, then there is an edge connecting two nodes in the coarse graph only if the distance between them in the original graph is at most 3.*

Proof. If there is an edge connecting j and j' in the coarse graph, then according to Theorem 1, there is an edge connecting $i \in \chi(j)$ and $i' \in \chi(j')$ in the original graph, for some nodes i and i'. Then by the assumption that the elements of $\chi(j)$ are either j or j's direct neighbors and similarly for $\chi(j')$, we know that j and j' are connected by the path $\{j, i, i', j'\}$, which means that the distance between j and j' is at most 3. □

Consequently, in what follows we will let S have the same sparsity structure as the corresponding part of $A + I$. The identity matrix is used to introduce self loops. An illustration of the resulting coarse graph is given in Figure 1(c), with self loops omitted.

Parameterization of S With the graph-theoretic interpretation of S, we now parameterize it. The strategy consists of the following computational steps. First, select coarse nodes in a differentiable manner, so that the sparsity structure of S is determined. Then, compute the nonzero elements of S.

The selection of coarse nodes may be done in several ways, such as the top-k approach that orders nodes by projecting their feature vectors along a learnable direction (see, e.g., Cangea et al. (2018); Gao and Ji (2019)). This approach, however, leverages only node features but not the graph information. To leverage both, we apply one graph convolution

$$\alpha = \text{sigmoid}(\widehat{A}XW_\alpha) \tag{2}$$

to compute a vector $\alpha \in \mathbb{R}^{n\times 1}$ that weighs all nodes (Lee, Lee, and Kang 2019). Here, $\widehat{A} \in \mathbb{R}^{n\times n}$ is the normalized graph adjacency matrix defined in graph convolutional networks (Kipf and Welling 2017), $X \in \mathbb{R}^{n\times d}$ is the node feature matrix, and $W_\alpha \in \mathbb{R}^{d\times 1}$ is a parameter vector. The weighting necessitates using sigmoid (or other invertible functions) rather than ReLU as the activation function.

For a coarsening into m nodes, we pick the top m values of α and list them in the sorted order. Denote by $\alpha_s \in \mathbb{R}^{m\times 1}$ such a vector, where the subscript s means sorted and

picked. We similarly denote by $\widehat{A}_s \in \mathbb{R}^{n\times m}$ the column-sorted and picked version of $\widehat{A}$.

We let S be an overlay of the graph adjacency matrix with the node weights α_s. Specifically, define

$$S = \ell_1\text{-row-normalize}[\widehat{A}_s \odot (\mathbf{1}\alpha_s^T)], \tag{3}$$

where $\mathbf{1}$ means a column vector of all ones.

There are several reasons why S is so defined. First, S carries the nonzero structure of $\widehat{A}_s$, which, following Corollary 2, renders more likely a sparse coarse graph. Second, the use of the normalized adjacency matrix introduces self loops, which ensure that an edge in the coarse graph exists if the distance is no more than three, rather than exactly three (which is too restrictive). Third, because both $\widehat{A}_s$ and α_s are nonnegative, the row normalization ensures that the total edge weight of the graph is preserved after coarsening. To see this, note that $\mathbf{1}^T A_c \mathbf{1} = \mathbf{1}^T S^T A S \mathbf{1} = \mathbf{1}^T A \mathbf{1}$.

Optimal Transport

The second ingredient of the proposed OTCOARSENING uses optimal transport for unsupervised learning. Optimal transport (Peyré and Cuturi 2019) is a framework that defines the distance of two probability measures through optimizing over all possible joint distributions of them. If the two measures lie on the same metric space and if the infinitesimal mass transportation cost is a distance metric, then optimal transport is the same as the Wasserstein-1 distance. In our setting, we extend this framework for defining the distance of the original graph G and its coarsened version G_c. Then, the distance constitutes the coarsening loss, from which model parameters are learned in an unsupervised manner.

Optimal Transport Distance To extend the definition of optimal transport of two probability measures to that of two graphs, we treat the node features from each graph as atoms of an empirical measure. The coarse node features result from graph neural network mappings, carrying information of both the initial node features and the graph structure. Hence, the empirical measure based on node features characterizes the graph and leads to a natural definition of graph distance.

Specifically, let M be a matrix whose element M_{ij} denotes the transport cost from a node i in G to a node j in G_c. We define the distance of two graphs as

$$W_\gamma(G, G_c) := \min_{P\in U(a,b)} \langle P, M\rangle - \gamma E(P), \tag{4}$$

where P, a matrix of the same size as M, denotes the joint probability distribution constrained to the space $U(a,b) := \{P \in \mathbb{R}_+^{n\times m} \mid P\mathbf{1} = a,\ P^T\mathbf{1} = b\}$ characterized by marginals a and b; E is the entropic regularization (Wilson 1969)

$$E(P) := -\sum_{i,j} P_{ij}(\log P_{ij} - 1);$$

and $\gamma > 0$ is the regularization magnitude.

Through a simple argument of Lagrange multipliers, it is known that the optimal P_γ that solves (4) exists and is unique, in the form $P_\gamma = \operatorname{diag}(u)K\operatorname{diag}(v)$, where u and v are certain positive vectors of matching dimensions and $K = \exp(-M/\gamma)$ with the exponential being element-wise. The solution P_γ may be computationally obtained by using Sinkhorn's algorithm (Sinkhorn 1964): Starting with any positive vector v^0, iterate

$$\text{for } i = 0, 1, 2, \ldots \text{ until convergence,}$$
$$u^{i+1} = a \oslash (Kv^i) \text{ and } v^{i+1} = b \oslash (K^T u^{i+1}). \tag{5}$$

Because the solution P_γ is part of the loss function to be optimized, we cannot iterate indefinitely. Hence, we instead define a computational solution P_γ^k by iterating only a finite number k times:

$$P_\gamma^k := \operatorname{diag}(u^k)K\operatorname{diag}(v^k). \tag{6}$$

Accordingly, we arrive at the k-step optimal transport distance

$$W_\gamma^k(G, G_c) := \langle P_\gamma^k, M\rangle - \gamma E(P_\gamma^k). \tag{7}$$

The distance (7) is the sample loss for training.

Parameterization of M With the distance defined, it remains to specify the transport cost matrix M. As discussed earlier, we model M_{ij} as the distance between the feature vector of node i from G and that of j from G_c. This approach on the one hand is consistent with the Wasserstein distance and on the other hand, carries both node feature and graph structure information.

Denote by $\mathrm{GNN}(A, X)$ a generic graph neural network architecture that takes the graph adjacency matrix A and node feature matrix X as input and produces as output a transformed feature matrix. We produce the feature matrix X_c of the coarse graph through the following encoder-decoder-like architecture:

$$Z = \mathrm{GNN}(A, X),\ Z_c = S^T Z,\ X_c = \mathrm{GNN}(A_c, Z_c). \tag{8}$$

The encoder produces an embedding matrix Z_c of the coarse graph through a combination of GNN transformation and aggregation S^T, whereas the decoder maps Z_c to the original feature space so that the resulting X_c lies in the same metric space as X. Then, the transport cost, or the metric distance, M_{ij} is the p-th power of the Euclidean distance of the two feature vectors:

$$M_{ij} = \|X(i,:) - X_c(j,:)\|_2^p. \tag{9}$$

In this case, the optimal transport distance is the p-th root of the Wasserstein-p distance. The power p is normally set as one or two.

Training and Downstream Use

With the technical ingredients developed in the preceding subsections, we summarize the computational steps into Algorithm 1, which is self explanatory.

After training, for each graph G we obtain a coarsening sequence and the corresponding node embedding matrices Z_c for each graph in the sequence. These node embeddings may be used for a downstream task. Take graph classification as an example. For each node embedding matrix, we

Algorithm 1 Unsupervised training: forward pass

for each coarsening level **do**
  Compute coarsening matrix S by (2) and (3)
  Obtain A_c and X_c by (1) and (8)
  Obtain also node embeddings Z_c from (8)
  Compute transport cost matrix M by (9)
  Compute k-step joint probability P_γ^k by (5) and (6)
  Compute current-level loss $W_\gamma^k(G, G_c)$ by (7)
  Set $G \leftarrow G_c$, $A \leftarrow A_c$, and $X \leftarrow X_c$
end for
Sum the loss for all coarsening levels as the sample loss

perform a global pooling (e.g., a concatenation of max pooling and mean pooling) across the nodes and obtain a summary vector. We then concatenate the summary vectors for all coarsening levels to form the feature vector of the graph. An MLP is then built to predict the graph label.

Experiments

In this section, we conduct a comprehensive set of experiments to evaluate the performance of the proposed method OTCOARSENING. Through experimentation, we aim at answering the following questions. (i) As an unsupervised hierarchical method, how well does it perform on a downstream task, compared with supervised approaches and unsupervised non-hierarchical approaches? (ii) In a multi-task setting, how well does it perform compared with supervised models trained separately for each task? (iii) Do the coarse graphs carry the structural information of the original graphs (i.e., are they meaningful)?

Setup

We perform experiments with the following data sets: PROTEINS, MUTAG, NCI109, IMDB-BINARY (IMDB-B for short), IMDB-MULTI (IMDB-M for short), and DD. They are popularly used benchmarks publicly available from Kersting et al. (2016). Except IMDB-B and IMDB-M which are derived from social networks, the rest of the data sets all come from the bioinformatics domain. Information of the data sets is summarized in Table 1.

Table 1: Data sets.

	PROTEINS	MUTAG	NCI109
# Graphs	1,113	188	4,127
# Classes	2	2	2
Ave. # nodes	39.06	17.93	29.68
Ave. node degree	3.73	2.21	2.17
	IMDB-B	IMDB-M	DD
# Graphs	1,000	1,500	1,178
# Classes	2	3	2
Ave. # nodes	19.77	13.00	284.32
Ave. node degree	9.76	10.14	5.03

We gauge the performance of OTCOARSENING with several supervised approaches. They include the plain GCN (Kipf and Welling 2017) followed by a gloabl mean pooling, as well as five more sophisticated pooling methods: SORTPOOL (Zhang et al. 2018), which retains the top-k nodes for fixed-size convolution; DIFFPOOL (Ying et al. 2018b), which applies soft clustering; SET2SET (Vinyals, Bengio, and Kudlur 2015), which is used together with GRAPHSAGE (Hamilton, Ying, and Leskovec 2017) as a pooling baseline in Ying et al. (2018b); GPOOL (Cangea et al. 2018; Gao and Ji 2019), which retains the top-k nodes for graph coarsening, as is used by GRAPH U-NET; and SAGPOOL (Lee, Lee, and Kang 2019), which applies self-attention to compute the top-k nodes. Among them, DIFFPOOL, GPOOL, and SAGPOOL are hierarchical methods, similar to ours. In addition, we also employ an ablation model, i.e., a supervised version of our coarsening model without using optimal transport distance as the loss function. This model is called OTCOARSENING-SUP, where we remove the unsupervised learning phase and directly train the coarsening model by using the prediction loss on training data.

Additionally, we take a simple unsupervised baseline. Named GRAPHAE-UNSUPV, this baseline is a graph autoencoder that does not perform coarsening, but rather, applies GCN twice to respectively encode the node features and decode for reconstruction. The encoder serves the same purpose as that of the plain GCN and the decoder is needed for training without supervised signals.

Experimentation Details

We evaluate all methods using 10-fold cross validation. For training, we use the Adam optimizer with a tuned initial learning rate and a fixed decay rate 0.5 for every 50 epochs. We perform unsupervised training for a maximum of 200 epochs and choose the model at the best validation loss. Afterward, we feed the learned representations into a 2-layer MLP and evaluate the graph classification performance.

The weighting vector α (cf. Equation (2)) used for coarse node selection is computed by using 1-layer GCN with activation function $\mathrm{sigmoid} \circ \mathrm{square}$. That is, $\alpha = \mathrm{sigmoid}((\widehat{A}XW_\alpha)^2)$. The GNNs in Equation (8) for computing the coarse node embeddings Z_c and coarse node features X_c are 1-layer GCNs. The power p in Wasserstein-p (cf. Equation (9)) is fixed as 2. We use grid search to tune hyperparameters: the learning rate is from $\{0.01, 0.001\}$; and the number of coarsening levels is from $\{1, 2, 3\}$ for the propoed method and $\{2, 3, 4\}$ for the compared methods. The coarsening ratio is set to 0.5 for all methods.

We implement the proposed method and the graph autoencoder by using the PyTorch Geometric library, which is shipped with off-the-shelf implementation of all other compared methods.

The code is available at https://github.com/matenure/OTCoarsening.

Graph Classification

Graph classification accuracies are reported in Table 2. OTCOARSENING outperforms the compared methods in five out of six data sets. Moreover, it improves significantly the

Table 2: Graph classification accuracy (in percentage).

Method	PROTEINS	MUTAG	NCI109	IMDB-B	IMDB-M	DD
GCN	72.3±3.1	73.4±5.2	69.6±1.3	71.3±5.2	50.5±2.4	71.8±4.1
SET2SET	73.4±3.7	74.6±5.3	70.3±1.6	72.9±4.7	49.7±3.5	70.8±3.9
SORTPOOL	73.5±4.5	80.1±6.7	69.1±4.5	71.6±3.6	49.9±2.1	73.7±7.7
DIFFPOOL	74.2±3.2	84.5±7.3	71.7±2.8	74.3±3.5	50.3±2.8	73.9±3.5
GPOOL	72.2±3.1	76.2±9.0	72.4±2.2	73.0±5.5	49.5±3.2	71.5±4.7
SAGPOOL	73.3±3.1	78.6±6.4	**73.1±2.4**	72.2±4.7	50.4±2.1	72.0±4.2
GRAPHAE-UNSUPV	74.3±3.6	84.6±8.0	66.4±4.6	72.4±5.9	49.9±2.9	76.5±2.8
OTCOARSENING-SUP	73.6±3.0	84.4±6.8	68.6±1.8	73.6±4.7	50.2±3.9	74.2±3.3
OTCOARSENING	**74.9±3.9**	**85.6±6.2**	68.5±5.2	**74.6±4.9**	**50.9±3.3**	**77.2±3.1**

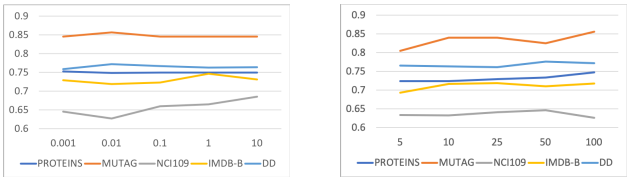
accuracy on DD over all supervised baselines. Interestingly, the supervised runner up is almost always DIFFPOOL, outperforming the subsequently proposed GPOOL and SAGPOOL. On the other hand, these two methods perform the best on the other data set NCI109, with SAGPOOL taking the first place. On this data set, OTCOARSENING performs on par with the lower end of the compared methods. It appears low-performing, possibly because of the lack of useful node features that play an important role in the optimal transport distance. Our ablation model, OTCOARSENING-SUP, is comparable to the best performance among all supervised baselines on most datasets, but performs worse than the final unsupervised model OTCOARSENING using optimal transport.

Based on these observations, we conclude that hierarchical methods indeed are promising for handling graph structured data. Moreover, as an unsupervised method, the proposed OTCOARSENING performs competitively with strong supervised approaches. In fact, even for the simple unsupervised baseline GRAPHAE-UNSUPV, it outperforms DIFFPOOL on PROTEINS, MUTAG, and DD. This observation indicates that unsupervised approaches are quite competitive, paving the way for possible uses in other tasks.

Sensitivity Analysis

OTCOARSENING introduces parameters owing to the computational nature of optimal transport: (a) the entropic regularization strength γ; and (b) the number of Sinkhorn steps, k. In Figure 2, we perform a sensitivity analysis and investigate the change of classification accuracy as these parameters vary. One sees that most of the curves are relatively flat, except the case of γ on NCI109. This observation indicates that the proposed method is relatively robust to the parameters of optimal transport. The curious case of NCI109 inherits the weak performance priorly observed, possibly caused by the lack of informative input features.

The observation that performance is insensitive to the parameters does not contradict the computational foundation of optimal transport. In the standard use, a transport cost M is given and the optimal plan P is computed accordingly. Hence, P varies with γ and k. In our case, on the other hand, M is not given. Rather, it is parameterized and the parameterization carries over to the computational solution of P. Thus, it is not impossible that the parameterization finds an optimum that renders the loss (Equation (7)) insensitive to γ and k. In other words, the optimization of Equation (7), with respect to neither M nor P but the parameters therein, turns out to be fairly stable.



(a) Entropic regularization, γ (b) Sinkhorn steps, k

Figure 2: Classification accuracy as parameters vary.

Multi-Task Learning

We further investigate the value of unsupervised graph representation through the lens of multi-task learning. We compare three scenarios: (A) a single representation trained without knowledge of the downstream tasks (method: OTCOARSENING, GRAPHAE-UNSUPV); (B) a single representation trained jointly with all downstream tasks (methods: GCN, SET2SET, SORTPOOL, DIFFPOOL, GPOOL, and SAGPOOL, all suffixed with "-joint"); and (C) different representations trained separately with each task (method: DIFFPOOL-sep).

The data set is QM7b (Wu et al. 2018), which consists of 14 regression targets. Following Gilmer et al. (2017), we standardize each target to mean 0 and standard deviation 1; we also use MSE as the training loss but test with MAE. Table 3 reports the MAE and timing results.

One sees from Table 3 that in terms of regression error, single unsupervised representation (A) significantly outperforms single supervised representations (B), whilist being inferior to separate supervised representations (C). Separate representations outperform single representations at the cost of longer training time, because they need to train 14 separate models whereas others only one. The timings for (B) are comparable with that of (A). The timing variation is caused by several factors, including the architecture difference and dense-versus-sparse implementation. DIFFPOOL is implemented with dense matrices, which may be faster compared with other methods that treat the graph adjacency matrix sparse, when the graphs are small.

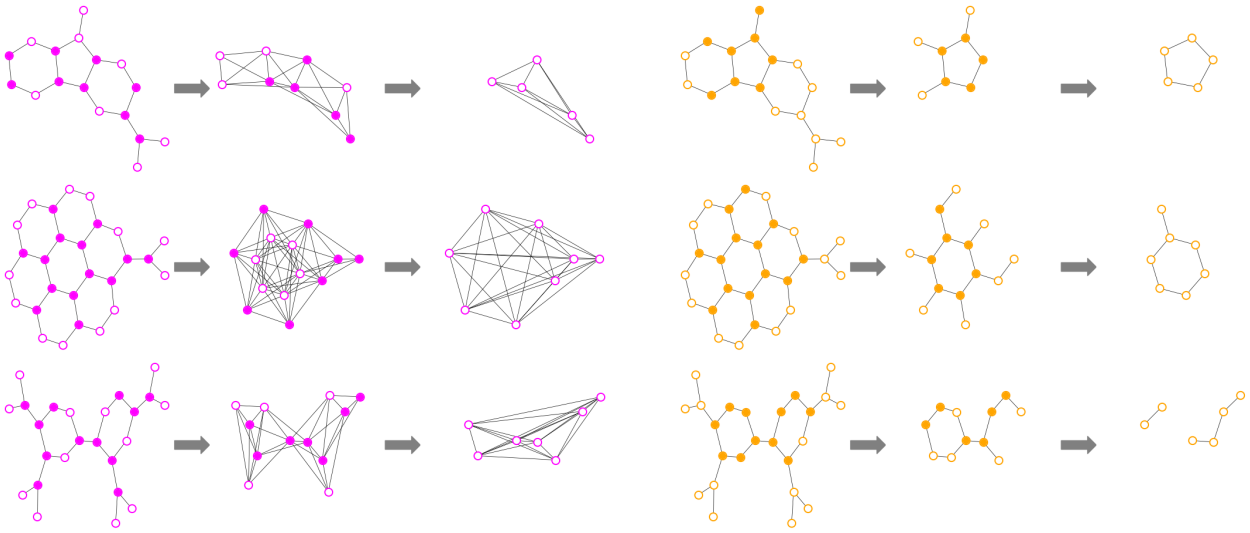


Figure 3: Coarsening sequence for graphs from MUTAG. Left (magenta): OTCOARSENING. Right (orange): SAGPOOL. Hollow nodes are coarse nodes.

Table 3: Multi-task regression error and training time (in seconds).

	Method	MAE	Time
(A)	OTCOARSENING	0.6625	1296
(A)	GRAPHAE-UNSUPV	0.6749	587
(B)	GCN-joint	2.4225	2122
(B)	SET2SET-joint	2.4256	2657
(B)	SORTPOOL-joint	2.4408	2652
(B)	DIFFPOOL-joint	2.4231	1100
(B)	GPOOL-joint	2.4200	2117
(B)	SAGPOOL-joint	2.4221	1874
(C)	DIFFPOOL-sep	0.1714	15520

Qualitative Study

As discussed in the related work section, coarsening approaches may be categorized in two classes: clustering based and node-selection based. Methods in the former class (e.g., DIFFPOOL) coarsen a graph through clustering similar nodes. In graph representation learning, similarity of nodes is measured by not only their graph distance but also the closeness of their feature vectors. Hence, two distant nodes bear a risk of being clustered together if their input features are similar.

On the other hand, methods in the latter class (e.g., GRAPH U-NET and SAGPOOL) use nodes in the original graph as coarse nodes. If the coarse nodes are connected based on only their graph distance but not feature vectors, the graph structure is more likely to be preserved. Such is the case for OTCOARSENING, where only nodes within a 3-hop neighborhood are connected. Such is also the case for GRAPH U-NET and SAGPOOL, where the neighborhood is even more restricted (e.g., only 1-hop neighborhood). However, if two coarse nodes are connected only when there is an edge in the original graph, these approaches bear another risk of resulting in disconnected coarse graphs.

Theoretical analysis is beyond scope. Hence, we conduct a qualitative study and visually inspect the coarsening results. In Figure 3, we show a few graphs from the data set MUTAG, placing the coarsening sequence of OTCOARSENING on the left and that of SAGPOOL on the right for comparison. The solid nodes are selected as coarse nodes.

For the graph on the top row, OTCOARSENING selects nodes across the consecutive rings in the first-level coarsening, whereas SAGPOOL selects the ring in the middle. For the graph in the middle row, both OTCOARSENING and SAGPOOL select the periphery of the honeycomb for the first-level coarsening, but differ in the second level in that one selects again the periphery but the other selects the heart. For the graph at the bottom row, OTCOARSENING preserves the butterfly topology through coarsening but the result of SAGPOOL is hard to comprehend.

Conclusion

Coarsening is a common approach for solving large-scale graph problems in various scientific disciplines. How one effectively selects coarse nodes and aggregates neighbors motivates the present work. Whereas a plethora of coarsening methods were proposed in the past and are used today, these methods either do not have a learning component, or have parameters that need be learned with a downstream task. In this work, we present OTCOARSENING, which is an unsupervised approach. It follows the concepts of AMG but learns the selection of the coarse nodes and the coarsening matrix through the use of optimal transport. We demonstrate its successful use in graph classification and regression tasks and show that the coarse graphs preserve the structure of the original one. We envision that the proposed idea may be adopted in many other graph learning scenarios and downstream tasks.

Acknowledgments

This work is supported in part by DOE Award DE-OE0000910.

References

Bravo-Hermsdorff, G.; and Gunderson, L. M. 2019. A Unifying Framework for Spectrum-Preserving Graph Sparsification and Coarsening. In *NeurIPS*.

Briggs, W. L.; Henson, V. E.; and McCormick, S. F. 2000. *A multigrid tutorial*. SIAM, second edition.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*.

Cangea, C.; Veličković, P.; Jovanović, N.; Kipf, T.; and Liò, P. 2018. Towards Sparse Hierarchical Graph Classifiers. In *NIPS Workshop on Relational Representation Learning*.

Chen, J.; Ma, T.; and Xiao, C. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

Chen, J.; and Safro, I. 2011. Algebraic Distance on Graphs. *SIAM Journal on Scientific Computing* 33(6): 3468–3490.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.

Dhillon, I.; Guan, Y.; and Kulis, B. 2007. Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(11): 1944–1957.

Duvenaud, D.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*.

Fey, M.; Lenssen, J. E.; Weichert, F.; and Müller, H. 2018. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. In *CVPR*.

Fout, A.; Byrd, J.; Shariat, B.; and Ben-Hur, A. 2017. Protein Interface Prediction using Graph Convolutional Networks. In *NIPS*.

Gao, H.; and Ji, S. 2019. Graph U-Nets. In *ICML*.

Garg, V. K.; and Jaakkola, T. 2019. Solving graph compression via optimal transport. In *NeurIPS*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep Convolutional Networks on Graph-Structured Data. arXiv:1506.05163.

Hendrickson, B.; and Leland, R. 1995. A Multi-Level Algorithm For Partitioning Graphs. In *SC*.

Jin, W.; Coley, C. W.; Barzilay, R.; and Jaakkola, T. 2017. Predicting Organic Reaction Outcomes with Weisfeiler-Lehman Network. In *NIPS*.

Karypis, G.; and Kumar, V. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* 20(1): 359–392.

Kernighan, B. W.; and Lin, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* 49: 291–307.

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark Data Sets for Graph Kernels. URL http://graphkernels.cs.tu-dortmund.de.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Kushnir, D.; Galun, M.; and Brandt, A. 2006. Fast multiscale clustering and manifold identification. *Pattern Recogn.* 39(10): 1876–1891.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-Attention Graph Pooling. In *ICML*.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2016. Gated Graph Sequence Neural Networks. In *ICLR*.

Liao, R.; Zhao, Z.; Urtasun, R.; and Zemel, R. 2019. LanczosNet: Multi-Scale Deep Graph Convolutional Networks. In *ICLR*.

Livne, O. E.; and Brandt, A. 2012. Lean Algebraic Multigrid (LAMG): Fast Graph Laplacian Linear Solver. *SIAM Journal on Scientific Computing* 34(4): B499–B522.

Loukas, A. 2019. Graph reduction with spectral and cut guarantees. *JMLR* .

Loukas, A.; and Vandergheynst, P. 2018. Spectrally approximating large graphs with smaller graphs. In *ICML*.

Luxburg, U. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17(4): 395–416.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In *AAAI*.

Maron, H.; Ben-Hamu, H.; Serviansky, H.; and Lipman, Y. 2019. Provably Powerful Graph Networks. In *NeurIPS*.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. In *AAAI*.

Peyré, G.; and Cuturi, M. 2019. Computational Optimal Transport. *Foundations and Trends in Machine Learning* 11(5–6): 355–607.

Ron, D.; Safro, I.; and Brandt, A. 2011. Relaxation-based Coarsening and Multiscale Graph Organization. *SIAM Journal on Multiscale Modeling and Simulation* 9: 407–423.

Ruge, J. W.; and Stüben, K. 1987. Algebraic Multigrid. In McCormick, S. F., ed., *Multigrid Methods*, Frontiers in Applied Mathematics, chapter 4. SIAM.

Safro, I.; Sanders, P.; and Schulz, C. 2014. Advanced coarsening schemes for graph partitioning. *Journal of Experimental Algorithmics* 19: 2.2:1–2.2:24.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20(1): 61–80.

Schütt, K. T.; Kindermans, P.-J.; Sauceda, H. E.; Chmiela, S.; Tkatchenko, A.; and Müller, K.-R. 2017. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *NIPS*.

Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* .

Shi, J.; and Malik, J. 2000. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8): 888–905.

Simonovsky, M.; and Komodakis, N. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *CVPR*.

Sinkhorn, R. 1964. A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices. *Annals of Mathematical Statistics* 35(2): 876–879.

Vayer, T.; Chapel, L.; Flamary, R.; Tavenard, R.; and Courty, N. 2019. Optimal Transport for structured data with application on graphs. In *ICML*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *ICLR*.

Vinyals, O.; Bengio, S.; and Kudlur, M. 2015. Order Matters: Sequence to sequence for sets. In *ICLR*.

Wang, M.; Tang, Y.; Wang, J.; and Deng, J. 2017. Premise Selection for Theorem Proving by Deep Graph Embedding. In *NIPS*.

Wilson, A. G. 1969. The Use of Entropy Maximising Models, in the Theory of Trip Distribution, Mode Split and Route Split. *Journal of Transport Economics and Policy* 3(1): 108–126.

Wu, Z.; Ramsundar, B.; Feinberg, E.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; and Pande, V. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chemical Science* 9(2): 513–530.

Xu, H.; Luo, D.; Zha, H.; and Carin, L. 2019a. Gromov-Wasserstein Learning for Graph Matching and Node Embedding. In *ICML*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019b. How Powerful are Graph Neural Networks? In *ICLR*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018a. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018b. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NIPS*.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*.