Distribution of Node Embeddings as Multiresolution Features for Graphs

Mark Heimann
University of Michigan
Ann Arbor, MI, USA
mheimann@umich.edu

Tara Safavi
University of Michigan
Ann Arbor, MI, USA
tsafavi@umich.edu

Danai Koutra
University of Michigan
Ann Arbor, MI, USA
dkoutra@umich.edu

***Abstract*—Graph classification is an important problem in many fields, from bioinformatics and neuroscience to computer vision and social network analysis. That said, the task of *comparing graphs* for the purpose of graph classification faces several major challenges. In particular, an effective graph comparison method must (1) expressively and inductively compare graphs; (2) efficiently compare large graphs; and (3) enable the use of fast machine learning models for graph classification.**

To address such challenges, we propose Randomized Grid Mapping (RGM), a fast-to-compute feature map that represents a graph via the distribution of its node embeddings in feature space. We justify RGM with close connections to kernel methods: RGM provably approximates the Laplacian kernel mean map and has the multiresolution properties of the pyramid match kernel. We also show that RGM can be extended to incorporate node labels using the Weisfeiler-Lehman framework. Extensive experiments show that graph classification accuracy with RGM feature maps is better than or competitive with many powerful graph kernels, unsupervised graph feature mappings, and deep neural networks. Moreover, comparing graphs based on their node embeddings with RGM is up to an order of magnitude faster than competitive baselines, while maintaining high classification accuracy.

I. Introduction

Graph classification has a wide range of applications from bioinformatics to computer vision to the social sciences. The problem can be solved with supervised machine learning given a suitable means of *graph comparison*. A strong and practical method for graph comparison must (**P1**) expressively and inductively compare graphs; (**P2**) efficiently handle many graphs with many non-aligned nodes; (**P3**) enable the downstream use of fast machine learning models for graph classification.

The first property (**P1**) implies that an ideal method should be flexible in characterizing graphs, and must handle unseen graphs. In terms of flexibility, many methods are constrained to compare topological graph properties via a small number of hand-engineered substructures. For example, most popular graph kernels are instances of the R-convolution framework, which decomposes a graph into substructures such as shortest paths, random walks, or graphlets, and compares graphs on the basis of these substructures [46]. Similarly, some works simply aggregate statistics (mean, standard deviation) of the distributions of hand-engineered node or edge features [3]. Moreover, not all methods readily generalize to out-of-sample nodes or graphs, often assuming that the graphs are defined on the same sets of vertices [22]. Likewise, optimal assignment kernels [25] are computed by inducing a hierarchy over the training and test data and are thus necessarily transductive.

Fig. 1: Overview of our framework. Given an input graph, node representations are learned via an appropriate embedding technique (§ II-B). Our proposed feature mapping RGM then aggregates the graph's node embeddings in vector space (§ III).

Furthermore, to perform graph classification effectively on large input graphs, it is necessary to compare graphs not only expressively but also efficiently (**P2**). This means that computing a graph feature representation or evaluating the kernel function between pairs of graphs must be scalable, ideally linear in the number of nodes across graphs. However, many existing feature representations are quadratic in the number of nodes [45], and R-convolution graph kernels can be even slower. For instance, the random walk graph kernel can take $O(n^3)$ time in the number of nodes across graphs [46].

While the domain-specific challenge to graph classification relies mainly on defining a means of graph comparison, the efficiency of the final graph classifier is also important (**P3**). For instance, graph kernels rely on comparatively slow kernel methods, for which specialized solvers take quadratic time or more in the number of inputs [17]. This limits their applicability to problems with large *numbers* of graphs. Meanwhile, deep neural networks often take many epochs to train and require specialized hardware. Unsupervised graph feature representations remain a practical, more scalable choice [44].

In this work, we propose **Randomized Grid Mapping** or RGM, a feature map for graphs that enjoys all of the desiderata mentioned above. RGM **characterizes each graph by the distribution of its node embeddings at multiple levels of resolution in vector space**, where node embeddings may be obtained from any unsupervised approach that generalizes across graphs. We justify RGM with novel theoretical connections to existing implicit kernels. RGM is flexible and capable of handling node labels within the powerful Weisfeiler-Lehman label expansion framework [40], making it highly expressive (**P1**). Moreover, RGM approximates an

TABLE I: Major symbols and definitions.

Symbol	Meaning
G_i	Graph $G_i = (V_i, E_i)$ with nodes V_i and edges E_i
n_i	Number of nodes in graph G_i
d	Node embedding dimensionality
$\mathbf{Y}_i$	Node embedding matrix for graph G_i in $\mathbb{R}^{n_i \times d}$
$\mathbf{y}_{i:j}$	Row-vector embedding in $\mathbb{R}^d$ of node j in graph G_i
$\boldsymbol{\delta}, \boldsymbol{\mu}$	Vectors in $\mathbb{R}^d$ of grid cell widths and offsets, resp.
$\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$	Random grid parametrized by cell width $\boldsymbol{\delta}$ and offset $\boldsymbol{\mu}$
$\mathbf{h}_i$	Histogram induced by grid $\mathcal{G}$ on G_i's embeddings $\mathbf{Y}_i$

implicit kernel in a fast, randomized fashion, leading to graph features that can be constructed in time linear in the number of nodes in that graph (**P2**). Finally, unlike exact kernel methods, RGM yields explicit features that can be used with linear SVMs [17] for scalable classification with many graphs (**P3**).

The contributions of this work include:

- *Feature mappings*. We propose Randomized Grid Mapping (RGM) feature maps, which characterize graphs by the distribution of their node embeddings at multiple levels of resolution. We generalize RGM to the Weisfeiler-Lehman label refinement scheme [40].
- *Theoretical analysis*. We justify RGM by proving that the dot product of its histograms of node embeddings approximate the Laplacian kernel mean map computed on sets of node embeddings between pairs of graphs. We also prove that we can extend our feature maps to more powerful composite kernels.
- *Extensive experiments*. Our experiments demonstrate that RGM achieves strong classification performance, efficiency, and scalability compared to a wide variety of competitive baselines including graph kernels, unsupervised feature representations, and deep neural networks.

The rest of this paper is structured as follows. In § II we give the preliminaries necessary to introduce RGM. In § III we propose and theoretically analyze RGM. In § IV we present an extensive range of experimental results. We discuss related work in § V, and offer concluding takeaways in § VI.

II. Preliminaries

We begin by outlining necessary background on embedding and kernel techniques for graph comparison. For reference, Table I gives our main symbols.

A. Problem definition and terminology

In graph classification, we are given a collection of training and test graphs of different sizes, with or without node labels. Each graph has a class that must be predicted. The i-th graph (in either the training or test set) is denoted $G_i = (V_i, E_i)$, where V_i and E_i are respectively the nodes and edges of graph G_i. We denote the number of nodes in G_i as $n_i \equiv |V_i|$.

Using node embedding, the graph G_i may be represented as a matrix $\mathbf{Y}_i \in \mathbb{R}^{n_i \times d}$ of d-dimensional vector embeddings. The vector embedding of node j in graph G_i is denoted $\mathbf{y}_{i:j} \in \mathbb{R}^d$. Without loss of generality, we assume embeddings are normalized to be in $[0, 1]^d$. Our goal is to train a machine learning hypothesis that can successfully predict the classes of the test graphs, given these embeddings.

B. Embedding techniques

Our proposed feature mapping RGM characterizes the distribution of a graph's node embeddings in latent feature space. While RGM can utilize any existing method for node embedding that inductively generalizes to *multi*-network settings, here we focus on three in particular. The first two have been previously used for graph classification [32], [53], and the third extends a structural node feature descriptor with subquadratic time complexity previously used for graph alignment [13]:

Eigenvector embeddings (EIG). Many graph similarity functions take as embeddings the eigenvectors of the adjacency matrix or the graph Laplacian, with node i represented by the absolute values of the i-th components of the top d eigenvectors [18], [32]. Eigenvectors capture *global* properties of the graph [32], which may generalize across graphs.

Return probability features (RPF). This method describes each node by a vector whose i-th entry represents the probability that an i-step random walk starting at that node returns to itself. This was shown to be an effective structural node feature descriptor [53], albeit requiring a full eigendecomposition of the adjacency matrix to compute exactly.

iNetMF. We extend the xNetMF [13] embedding technique, which was originally used for multi-network alignment. At a high level, xNetMF constructs a histogram per node that captures the degree distribution in that node's (weighted) k-hop neighborhood. xNetMF efficiently computes embeddings using these histograms by comparing each node to a small sample of landmark nodes randomly chosen from all training graphs, then constructing a low-rank implictit factorization of a structural node similarity matrix leveraging the Nyström method [6]. We make xNetMF inductive (i.e., **iNetMF**) by reusing the same "landmark nodes" from the training set to embed the test graphs, in effect embedding the test graphs into the same subspace as the training graphs. This can be viewed as a simplified version of latent network summarization [16] using the fast Nyström decomposition.

C. Embedding kernel techniques

We briefly overview existing kernel methods that operate on the node embeddings of pairs of graphs, accepting node embedding matrices $\mathbf{Y}_1 \in \mathbb{R}^{n_1 \times d}$ and $\mathbf{Y}_2 \in \mathbb{R}^{n_2 \times d}$ for graphs G_1 and G_2, respectively, as input. Embeddings may be compared in two different ways:

Distance-based. One way to compare G_1 and G_2 is to compute the (continuous) distances between pairs of their node embedding vectors. While many such comparison methods exist, such as the Earth Mover's Distance (used in [32]), here we focus on the **Laplacian kernel mean map** [41], which,

for embedding matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$ and hyperparameter γ controlling the kernel's resolution, is

$$k_{\text{LKM}}(\mathbf{Y}_1, \mathbf{Y}_2; \gamma) = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \exp\left(-\gamma ||\mathbf{y}_{1:i} - \mathbf{y}_{2:j}||_1\right). \quad (1)$$

Equation (1) corresponds to the average Laplacian kernel similarity between all pairs of embeddings in graphs G_1 and G_2 and has $O(n_1 n_2 d)$ complexity to compute for d-dimensional node embeddings.

Grid-based. An alternative (discretized) approach is to compute the embeddings' spatial overlap on a grid or histogram. Here we focus on the pyramid match or **PM kernel** [8], which fits a set of increasingly finer resolution grids to the d-dimensional unit hypercube. As used in graph classification [32], the grid at each level $\ell \in 0, \ldots, L$ has 2^ℓ cells of equal width without offset from the origin. At each level ℓ, these grids induce histograms $\mathbf{h}_1^{(\ell)}$ and $\mathbf{h}_2^{(\ell)}$ capturing the number of node embeddings from $\mathbf{Y}_1$ and $\mathbf{Y}_2$ that map into each grid cell. The intersection of pairs of histograms at level ℓ across cells c is given as $I(\mathbf{h}_1^{(\ell)}, \mathbf{h}_2^{(\ell)}) = \sum_c \min\{h_{1,c}^{(\ell)},\ h_{2,c}^{(\ell)}\}$. The PM kernel, which takes $O((n_1 + n_2)dL)$ time to compute, is a weighted sum of new intersections found at each increasingly coarse grid:

$$k_{\text{PM}}(\mathbf{Y}_1, \mathbf{Y}_2; L) = \sum_{\ell=0}^{L-1} \frac{1}{2^{L-\ell}} \left[I\big(\mathbf{h}_1^{(\ell)}, \mathbf{h}_2^{(\ell)}\big) - I\big(\mathbf{h}_1^{(\ell+1)}, \mathbf{h}_2^{(\ell+1)}\big) \right]. \quad (2)$$

III. RGM FEATURE MAPS

With the necessary background given, we now discuss how we aggregate a collection of node embeddings into a unified *explicit* feature map for a graph. We first propose our histogram-based mapping RGM and prove its connection to the Laplacian kernel mean map. We then generalize RGM to a multiresolution feature map, and further extend it to incorporate node labels within the Weisfeiler-Lehman framework.

A. Randomized features of graphs

In this section we propose our feature mapping RGM, and theoretically justify it by connecting it to the existing kernel techniques discussed in § II-C.

Histograms of node embeddings. RGM builds on the intuition of grid-based binning (§ II-C) for a fast-to-compute *feature mapping* that can be used with linear SVMs for efficient graph classification. Let $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ be a *randomized grid* specified by d-dimensional random vectors $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$, which respectively specify the cell width and offset of the grid along each dimension. Given graph G_i with node embedding matrix $\mathbf{Y}_i$, with a hash function $\phi(\cdot)$ mapping each node's embedding to a cell in $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ we induce a histogram $\mathbf{h}_i$. The value of the j-th element of $\mathbf{h}_i$ is

$$h_{i,j} = \sum_{p=1}^{n_i} \mathbb{1}\left\{ \phi\big(\lceil (\mathbf{y}_{i:p} - \boldsymbol{\mu})/\boldsymbol{\delta} \rceil\big) = j \right\}. \quad (3)$$

We can use $\mathbf{h}_i$ as a *feature vector* for graph G_i. Intuitively, each cell in the histogram represents a region of d-dimensional embedding space, so these features count the number of embeddings that fall into each region of the space. In other words, we describe G_i in terms of the *distribution of its node embeddings* in vector space.

Probabilistic interpretation. Our randomized grid construction, given a suitable choice of parameters $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$, gives RGM a probabilistic kernel interpretation. Specifically, the dot product of two graphs' RGM histograms approximates the Laplacian kernel mean map between the graphs. We first state a foundational result about random features for general kernel methods:

Lemma 1 (Adapted from [35])**.** *For vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, the probability that $\mathbf{x}_1$ and $\mathbf{x}_2$ map to the same cell in random grid $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ with cell widths $\boldsymbol{\delta}_i$ drawn from a Gamma distribution with shape 2 and scale $\frac{1}{\gamma}$, and offsets $\boldsymbol{\mu}_i \sim$ Uniform$(0, \delta_i)$ sampled independently along each dimension i is equal to the Laplacian kernel $\exp(-\gamma ||\mathbf{x}_1 - \mathbf{x}_2||_1)$.*

Using this lemma, we connect the embedding histograms of Equation (3) and the Laplacian kernel mean map:

Theorem 1. *Let $\mathbf{h}_1$ and $\mathbf{h}_2$ be the normalized histograms induced via* RGM *on graph node embedding matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$ respectively, by a grid $\mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}$ with random cell widths $\boldsymbol{\delta}_i$ drawn from a gamma distribution with shape 2 and scale $\frac{1}{\gamma}$, and offsets $\boldsymbol{\mu}_i \sim$ Uniform$(0, \delta_i)$ sampled independently along each dimension i. Then*

$$\mathbb{E}\Big[\langle \mathbf{h}_1, \mathbf{h}_2 \rangle\Big] = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \exp(-\gamma ||\mathbf{y}_{1:i} - \mathbf{y}_{2:j}||_1),$$

where the right-hand side is equivalent to the Laplacian kernel mean map (1) *between embedding matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$.*

Proof. For each node $i \in V_1$, let $\mathbf{f}_i$ be a binary indicator vector with $f_{ic} = \mathbb{1}\{\mathbf{y}_{1:i} \in \mathcal{G}^{[\boldsymbol{\delta},\boldsymbol{\mu}]}[c]\}$; i.e., 1 if i's embedding falls into grid cell c. We define the indicator vectors for V_2 similarly. Then we have that

$$\langle \mathbf{h}_1, \mathbf{h}_2 \rangle = \frac{1}{n_1 n_2} \sum_{c \in \mathcal{G}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ic}\, f_{jc} = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{f}_i^\top \mathbf{f}_j,$$

since the product of the numbers of nodes in G_1 and G_2 that fall into the same cell c is the number of the corresponding cross-graph node pairs. This can be determined by multiplying the corresponding indicator vectors for all these pairs, since the product will be 0 when the nodes do not fall into the same cell. Recall that the vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ depend on the parameter γ governing the distribution from which the components of $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$ are sampled. Their dot product is 1 iff the embeddings of node i in G_1 and node j in G_2 map to the same cell in $\mathcal{G}$. Thus, $\mathbb{E}[\mathbf{f}_i^\top \mathbf{f}_j; \gamma] = \exp(-\gamma ||\mathbf{y}_{1:i} - \mathbf{y}_{2:j}||_1)$, and the theorem follows from Lemma 1. □

This result offers a theoretical connection between our feature maps based on node embedding distributions and graph kernels, namely the Laplacian kernel mean map. Indeed, we see a new connection between distance-based and grid-based embedding comparison techniques (§ II-C): with appropriate

grid construction, the latter can be used to approximate the former, in linear time in the number of nodes in each graph. An important advantage that our explicit feature maps have over both kernels is that faster linear machine learning algorithms may be used, which scale better to large numbers of graphs.

B. Multiresolution feature maps

Representing each graph using embedding histograms from Equation (3) with grid construction as in Theorem 1 allows us to construct a feature map that approximates the Laplacian kernel mean map for a particular resolution given by a fixed γ. A large value of γ drives the kernel similarity function closer to zero, meaning only extremely similar nodes will contribute meaningfully to the kernel mean map. Meanwhile, a small value drives the kernel similarity function close to one, in which case even rather dissimilar nodes may still measure a relatively high similarity.

Composite kernels and composite feature maps. Any single kernel or parametrization has strengths and drawbacks, and a feature map that approximates that single kernel shares that kernel's limitations. A powerful and arguably more flexible technique, then, is to create *composite* kernels from linear combinations of single kernels. Defining α_i as the contribution of the i-th kernel k_i, composite kernels have the form

$$K(\mathbf{Y}_1, \mathbf{Y}_2) = \sum_{i=1}^{M} \alpha_i k_i(\mathbf{Y}_1, \mathbf{Y}_2). \tag{4}$$

Similarly, we can create composite feature maps with similarly greater expressive power.Specifically, we show that if the individual kernels comprising a composite kernel are approximable by random features, we have a (random) feature map for the corresponding composite kernel:

Lemma 2. *Given kernels $k_1(\mathbf{Y}_1, \mathbf{Y}_2), \ldots, k_M(\mathbf{Y}_1, \mathbf{Y}_2)$ with approximate feature maps, the composite kernel $K = \sum_{i=1}^{M} \alpha_i k_i(\mathbf{Y}_1, \mathbf{Y}_2)$ (i.e., Equation (4)) has a corresponding approximate feature map.*

Proof. Let $\psi_i(\cdot)$ be a function that, for embeddings $\mathbf{Y}_i$, constructs features approximating the individual kernel k_i: $k_i(\mathbf{Y}_1, \mathbf{Y}_2) \approx \psi_i(\mathbf{Y}_1)^\top \psi_i(\mathbf{Y}_2)$. We define the feature map for embedding $\mathbf{Y}_i$ as $\mathbf{h}_i = [\sqrt{\alpha_1}\psi_1(\mathbf{Y}_i) \,||\, \ldots \,||\, \sqrt{\alpha_M}\psi_M(\mathbf{Y}_i)]$, where $||$ denotes vector concatenation. Then

$$\begin{aligned} \mathbf{h}_1^\top \mathbf{h}_2 &= \sum_{i=1}^{M} \alpha_i \psi_i(\mathbf{Y}_1)^\top \psi_i(\mathbf{Y}_2) \\ &\approx \sum_{i=1}^{M} \alpha_i k_i(\mathbf{Y}_1, \mathbf{Y}_2) = K(\mathbf{Y}_1, \mathbf{Y}_2). \end{aligned}$$

□

Multiresolution RGM *features*. With Lemma 2, we now have the tools to develop our RGM features based on histograms of node embeddings that overcome the limitations of any fixed resolution by combining *multiple* levels of resolution. That is, by *concatenating* node embedding histograms across L levels of resolution, we achieve the effect of a composite Laplacian kernel mean map with different values of γ.

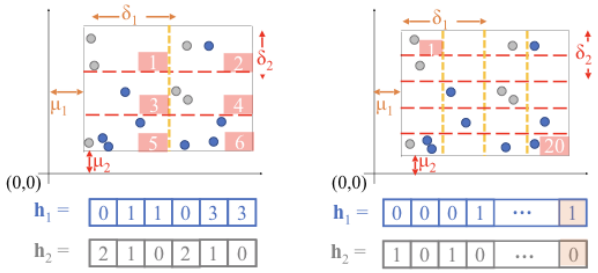


Fig. 2: Multiresolution feature maps for graphs. We create histograms by binning a graph's node embeddings using grids with randomly chosen cell widths and offsets along each dimension. We use multiple grids parametrized differently in expectation to produce histograms of coarser (left) and finer (right) levels of resolution. The final graph features are a weighted concatenation of these histograms.

At each level of resolution $\ell \in [0, 1, \ldots, L]$, we construct component histograms $\mathbf{h}_i^{(\ell)}$ from $\mathbf{Y}_i$ using Equation (3), with cell widths drawn from a gamma distribution with shape 2 and scale $\frac{1}{2^{\ell+1}}$ along with uniform offsets (recall that the scale corresponds to the inverse of γ in the Laplacian kernel mean map, by Theorem 1). The expected cell width along each dimension for $\mathbf{h}_i^{(\ell)}$ is $\frac{1}{2^\ell}$. The earlier histograms will thus have coarse cells that capture many matches, while later histograms will have fine cells that only bin together embeddings very close in vector space, as demonstrated in Figure 2.

As in [32], we use a weighing scheme to prioritize matches found at more discriminative finer resolutions: a histogram with expected cell width $\frac{1}{2^\ell}$ has weighing factor $\sqrt{1/2^{L-\ell}}$. The dot product of two component histograms with this weighing factor will then be weighed by $1/2^{L-\ell}$. Putting it all together, for a graph G_i with node embeddings $\mathbf{Y}_i$, our RGM feature map for a set of node embeddings is

$$\mathbf{h}_i = [\sqrt{1/2^L}\mathbf{h}_i^{(0)} \,||\, \sqrt{1/2^{L-1}}\mathbf{h}_i^{(1)} \,||\, \ldots \,||\, \mathbf{h}_i^{(\ell)}] \tag{5}$$

This multiresolution design recalls the design of the pyramid match kernel (§ II-C) while retaining the theoretical connections to the Laplacian kernel mean map discussed in the previous section. However, it should be noted that the multiresolution RGM is *not* approximating the PM kernel, but rather has a similar design that compares graphs at multiple levels of resolution in vector space. Two key differences between multiresolution RGM and PM are: (1) PM compares embeddings via histogram intersection versus RGM's dot product, and (2) PM excludes nodes matched at finer levels of granularity before comparing coarser levels of granularity. Concerning the former, RGM's dot product permits the use of faster (linear) machine learning algorithms. Concerning the latter, by including matches in all levels, RGM places further weight on matches found in fine levels of granularity, which are likely to be matched at coarser levels of granularity as well, amplifying the effect that PM attempts to achieve.

Complexity analysis. Each level of RGM hashes n_i nodes per graph G_i, and each graph is represented by features of d

dimensions. Therefore, a single level of RGM is $O(n_i d)$. With L total levels of resolution, RGM's complexity is $O(n_i dL)$.

C. Handling node labels

Node labels may provide an additional source of information beyond the graph topology alone. Without loss of generality, it suffices to consider discrete node labels, as continuous attributes may be hashed into discrete labels [29]. We review techniques that transform unlabeled graph kernels into labeled ones and make simple labeled kernels more powerful. For each technique, we show that using Lemma 2, RGM feature maps can have equivalent capabilities.

Composite labeled kernels. Given a kernel between sets of embeddings $k(\mathbf{Y}_1, \mathbf{Y}_2)$, a corresponding composite *labeled* kernel is

$$K_B(\mathbf{Y}_1, \mathbf{Y}_2) = \sum_{b \in B} k(\mathbf{Y}_1^{\{b\}}, \mathbf{Y}_2^{\{b\}}), \tag{6}$$

where B is the set of unique node labels, $\mathbf{Y}_i^{\{b\}}$ consists of the embeddings in G_i of nodes with label b, and k is the (unlabeled) base kernel, such as the pyramid match kernel [32], or our multiresolution weighted sum of Laplacian kernel mean maps approximated by RGM. Intuitively, the idea is to use the base kernel to only compare nodes with the same label.

We follow this intuition to design labeled features by forming multiresolution histograms using Equation (5) for embeddings of nodes with each label and concatenating them. From Lemma 2, it follows that this labeled version of RGM corresponds to the labeled kernel built on the (multiresolution) Laplacian kernel mean map using Equation (6).

Corollary 1. *Given graph G_i with embedding matrix $\mathbf{Y}_i$, the feature map*

$$\mathbf{h}_i = [\mathbf{h}_i^{\{b_1\}} \,||\, \ldots \,||\, \mathbf{h}_i^{\{b_{|B|}\}}] \tag{7}$$

approximates the labeled Laplacian kernel mean map.

Each $\mathbf{h}_i^{\{b\}}$ refers to an RGM feature map constructed using Equation (5) for nodes with label b only, with embeddings $\mathbf{Y}_i^{\{b\}}$. As each node is still mapped to only one cell in the corresponding grid, the worst-case complexity of RGM is unchanged. Thus, we can maintain linear-time feature construction and training *even with node labels* using RGM.

RGM *with Weisfeiler-Lehman framework*. We can further generalize the labeled feature maps from Equation (7) to the Weisfeiler-Lehman (WL) framework [40], a state-of-the-art graph kernel framework that over H iterations assigns each node a new label by hashing its neighbors' labels in the previous iterations. Given a labeled graph kernel $K_B(\mathbf{Y}_1, \mathbf{Y}_2)$ as in Equation (6), the corresponding WL kernel is

$$K_{\text{WL}}(\mathbf{Y}_1, \mathbf{Y}_2; H) = \sum_{h=0}^{H} K_{B_h}(\mathbf{Y}_1, \mathbf{Y}_2), \tag{8}$$

where for H iterations, B_h is the Weisfeiler-Lehman labeling at iteration h, and B_0 is the set of original node labels. In the above, we sum individual kernels that use the WL labelings at each iteration. Thus, applying Lemma 2 and Corollary 1, we design a version of RGM corresponding to the labeled Laplacian kernel mean map enhanced with the WL framework:

Corollary 2. *Given graph G_i with embedding matrix $\mathbf{Y}_i$, the feature map $\mathbf{h}_i = [\mathbf{h}_i^{\{B_0\}} \,||\, \ldots \,||\, \mathbf{h}_i^{\{B_H\}}]$ is an approximate feature map for the H-iteration Weisfeiler-Lehman Laplacian kernel mean map, where B_h is the WL labeled at iteration h and B_0 is the original set of node labels.*

Here, the component histograms $\mathbf{h}_i^{\{B_h\}}$ that we concatenate for each relabeling are constructed using Equation (7).

Complexity analysis. WL RGM is $O(n_i dLH)$ for H label expansions. Therefore, by designing linear feature maps to approximate WL graph kernels using node embeddings, we can use the well-documented strengths of WL label expansion [40] to achieve good performance *faster than exact kernel methods*.

TABLE II: Real data [19] used in our experiments. We give the total number of nodes/edges across all graphs per dataset.

Name	Nodes	Edges	Graphs	Classes	Node labels	Domain
MUTAG	3 371	3 721	188	2	Y	bioinf
PTC(-MR)	4 916	5 053	344	2	Y	bioinf
NCI1	122 765	132 753	4 110	2	Y	bioinf
IMDB (binary)	19773	96531	1000	2	N	collab
IMDB (multi)	19 502	98 910	1 500	3	N	collab
COLLAB	372 474	12 286 733	5000	3	N	collab

IV. Experiments

We now study RGM across a range of extensive experiments. We focus on the following research questions:

- **Q1** How accurately can we classify graphs with RGM feature maps?
- **Q2** How efficient and scalable is RGM relative to related kernel methods with respect to the number and/or size of the input graphs?
- **Q3** Can other node embedding or aggregation choices be used in RGM, particularly in an inductive setting?

A. Experimental setup

We evaluate our methods on six benchmark graph classification datasets from different domains commonly studied in graph classification–bioinformatics and social collaboration–all publicly available with detailed descriptions at [19]. Table II presents aggregate information about each dataset.

Embedding methods. As discussed in § II-B, we use three embedding methods with RGM feature maps:

1) **EIG**: Following [32], we take the top 6 eigenvectors of the adjacency matrix to form the embeddings (if a graph has size $n < 6$, we repeat the last features $6 - n$ times).
2) **RPF**: To compute the return probability features, we use the recommended $d = 50$ [53].
3) **iNetMF**: We set the maximum hop distance $K = 2$ and discount factor $\delta = 0.1$, following [13], with embedding dimensionality $d = 100$, as per the literature.

For brevity, in our results we report RGM's performance with the most accurate embedding method for each dataset among EIG, RPF, and iNetMF. In general, they perform comparably across datasets.

Baselines. We compare RGM against several popular baselines from the **graph kernel literature**:

1) **SP** [5], or the shortest paths kernel;
2) **GR** [38], or the graphlets kernel. We follow the literature and using graphlets of size 3 [32];
3) **WL-ST** [40], or the Weisfeiler-Lehman subtree kernel;
4) **WL-OA** [25], or the Weisfeiler-Lehman optimal assignment kernel;
5) **LWL-3** [28] kernel;
6) **WL-PM** [32], which computes the Weisfeiler-Lehman pyramid match kernel on eigenvector embeddings;
7) **RetGK** [53], a graph kernel based on the return probabilities of random walks as captured by RPF. We use RetGK_{II}, which uses approximate random features techniques to avoid a quadratic-time comparison of graphs using RPF and thus conceptually resembles our approach.

From the **unsupervised feature mapping** literature, we compare to:

a) **NetLSD** [44], which achieved superior performance and scalability over unsupervised feature representations such as NetSimile [3] and FGSD [45]. We use both the heat and the wave kernel to obtain graph representations, and report the best results for each dataset.

Finally, following existing practice [53], we compile reported numbers for **deep neural networks** for further comparison:

i) **DCNN** [2], or diffusion-convolutional neural networks;
ii) **PSCN** [31], or the PATCHY-SAN neural network;
iii) **DCGNN** [52], a neural architecture that performs end-to-end graph classification.

Note that NetLSD, SP, and GR do not use node labels, while the other baselines do. For datasets without node labels, we give all nodes the same label to start [25]. We fix the number of WL iterations for RGM and WL baselines to $H = 2$ [24] and the number of levels in PM and RGM to 4 [32]. Other parameters specific to particular baseline methods are set to values recommended by their authors in the papers and/or official implementations.

We used MATLAB public implementations of the SP, GR, and WL-ST baselines [39]. We used the official implementations of NetLSD, LWL-3, and RetGK written in Python, C++, and MATLAB respectively, as well as a MATLAB implementation of WL-OA from the authors of the paper [25]. We implemented the PM kernel following [32] in Python, along with RGM. Our code is available at https://github.com/GemsLab/RGM. All experiments ran on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz with 256GB RAM.

B. How accurate is RGM?

Task. We perform 10-fold cross validation averaged over five trials and report the average accuracy and standard deviation.We use a linear SVM to classify feature mappings and a kernel SVM classifier for kernel matrices, all from scikit-learn [33], limiting the solver to 10^4 iterations and choosing the SVM parameter C by cross-validation from $\{10^{-3}, 10^{-2}, \ldots, 10^3\}$.

Results. We report graph classification accuracy over all baselines and RGM in Table III. RGM yields highly competitive performance against existing graph kernels. It is the most accurate method on two datasets: the most of any method, tied only with WL-PM. The only dataset where WL-PM outperforms RGM significantly is COLLAB, as it only ekes ahead on PTC-MR. However, RGM outperforms WL-PM significantly on both NCI1 and IMDB-M. Moreover, RGM is never lower than fourth best out of all the baselines on each dataset: a consistent performance (all other baselines besides WL-ST and WL-PM finish in the bottom half at least once).

Compared to the recent feature representation NetLSD, RGM is more accurate on all datasets under consideration. One reason for this may be that NetLSD does not use node labels. Even on datasets that do not have node labels (the three collaboration datasets), the Weisfeiler-Lehman framework can be used to generate meaningful label expansions that RGM can capitalize on but NetLSD cannot.

Finally, compared to published results from recent and widely used deep neural network methods, RGM performs highly favorably. It is more accurate than all of them on almost all datasets, in many cases (NCI, COLLAB) by a wide margin. One note is that on the smallest datasets MUTAG and PTC, we see extremely high variances especially for PSCN. Many deep learning models for graph classification have been noted [52] to overfit on smaller datasets in particular, which is one of the practical difficulties of training them.

Observation 1. RGM *is among the most accurate methods for graph classification, compared to a variety of powerful recent baselines. It is competitive with leading techniques from all three major areas of graph classification literature: unsupervised feature learning, kernels, and deep neural networks.*

C. How efficient is RGM?

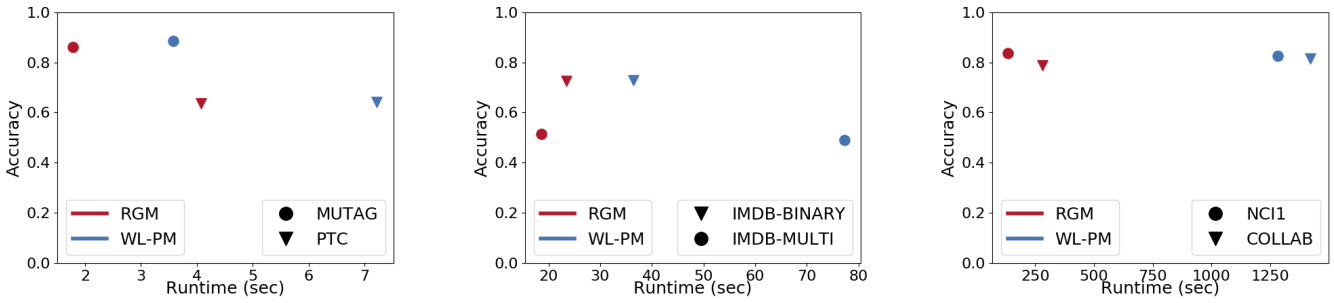
We now focus on the runtime of RGM, as this is a significant practical benefit afforded by explicit feature maps compared to many other methods such as kernels. Here we focus on the pyramid match kernel, which is the most related baseline both conceptually and in terms of results (Table III).

Task. We study the accuracy versus runtime taken to compare embeddings using RGM and WL-PM on our six benchmark datasets. We group the two smallest datasets (the bioinformatics datasets MUTAG and PTC), the two medium-size datasets (the two IMDB datasets), and the two largest datasets (the NCI bioinformatics dataset and the COLLAB dataset) together so that the plots include comparable magnitudes of runtime.

Results. In Figure 3, we see that not only does RGM lead to highly *accurate* graph classification, its *runtime* is favorable compared to implicit kernel methods that must compute and manipulate a quadratic kernel matrix. The speedup afforded by

TABLE III: Accuracy of RGM versus graph kernels, feature learning algorithms, and deep neural networks. We see that RGM is one of the most accurate methods on all datasets, compared to baselines from many different fields. (*: Results reported from original papers. For DCNN, we report results, which did not include standard deviations, from the original paper [2] on datasets used in that paper. We report the remaining results from [52]. >12hr means that computation was not finished within 12 hours.)

Method	MUTAG	PTC	NCI1	IMDB-BINARY	IMDB-MULTI	COLLAB
DCNN*	67.0	55.3	62.6	49.1 ± 1.37	33.5 ± 1.42	52.1 ± 0.71
PSCN*	89.0 ± 4.37	62.3 ± 5.68	76.3 ± 1.68	71.0 ± 2.29	45.2 ± 2.84	72.6 ± 2.15
DGCNN*	85.8 ± 1.66	58.6 ± 2.47	74.4 ± 0.47	70.0 ± 0.86	47.8 ± 0.85	73.8 ± 0.49
NETLSD	82.9 ± 0.58	58.7 ± 1.06	62.6 ± 0.25	64.6 ± 0.39	45.9 ± 1.04	66.7 ± 0.11
GR	83.1 ± 0.77	56.7 ± 0.65	62.8 ± 0.08	55.1 ± 0.83	37.0 ± 1.99	60.4 ± 0.08
SP	88.2 ± 0.24	57.6 ± 0.49	66.2 ± 0.44	51.9 ± 1.31	35.4 ± 1.08	43.0 ± 3.27
WL-ST	86.3 ± 1.13	63.0 ± 1.54	82.2 ± 0.19	72.3 ± 0.35	47.7 ± 0.55	78.4 ± 0.15
LWL3	84.0 ± 1.14	58.8 ± 1.52	77.8 ± 2.12	72.3 ± 0.63	46.0 ± 1.22	>12hr
WL-OA	86.0 ± 0.82	62.2 ± 1.10	82.9 ± 0.23	73.3 ± 0.15	48.2 ± 1.04	80.6 ± 0.29
RETGK	86.3 ± 1.22	61.4 ± 0.87	80.7 ± 0.19	72.6 ± 0.83	45.5 ± 0.79	80.8 ± 0.32
WL-PM	88.4 ± 1.10	64.1 ± 0.52	82.6 ± 0.21	73.0 ± 0.48	49.1 ± 0.73	81.5 ± 0.35
RGM	87.8 ± 1.05	63.6 ± 1.53	83.7 ± 0.19	73.0 ± 1.04	51.5 ± 0.40	78.6 ± 0.13



(a) Small datasets: MUTAG & PTC **(b) Mid-size datasets: IMDB-B & IMDB-M** **(c) Large datasets: NCI & COLLAB**

Fig. 3: Upper left quadrant is best: Accuracy vs runtime for RGM and its closest competitor WL-PM. We denote datasets by marker shape and methods by color. Across all sizes of datasets, RGM has comparable accuracy and considerably faster runtime.

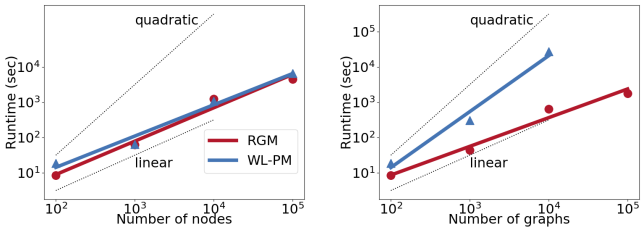
RGM is apparent on all sizes of datasets, but is particularly noticeable on large datasets. Meanwhile, accuracy is very comparable, as also seen in Table III.

Observation 2. RGM *achieves a favorable balance of accuracy and speed compared to exact kernel methods.*

We further illustrate this point by constructing large datasets and studying the scalability of the two methods as the number or size of the graphs increases in a controlled manner.

Task. To evaluate the scalability of RGM compared to PM, we measure both methods' runtime for graph classification based on comparing embeddings of increasingly large Erdős-Rényi graphs with random binary labels. We use eigenvector embeddings for RGM as well as PM and do not use WL label expansion. For our first experiment, the datasets consist of 100 graphs of 100-100K *nodes* each. In the second experiment, the datasets consist of 100-100K *graphs* of 100 nodes each.

Results. We plot the runtime averaged over five independent trials in Figure 4. In Figure 4a, we see that both methods scale approximately linearly with the number of nodes in the input graphs, as their asymptotic complexities suggest. However, in Figure 4b, the kernel-based classifier used by PM is much slower than the linear SVM that can be used with RGM.



(a) Wrt number of nodes **(b) Wrt number of graphs**

Fig. 4: Scalability of RGM. Dotted linear and quadratic slopes plotted for reference. RGM scales linearly with respect to both the *number* and *size* of the input graphs. In contrast, the PM kernel does not scale with the number of graphs.

Indeed, we cannot even compute the quadratic 100K by 100K kernel matrix for PM within 12 hours. However, RGM finishes well within this timeframe on 100K graphs, and is indeed faster for all numbers of graphs we consider in this experiment. We see that it scales approximately linearly with the number of graphs, in accordance with its asymptotic complexity [17].

Observation 3. RGM *is an efficient method for graph comparison and classification, scaling linearly in both the* number *and the* size *of graphs. It can be used on datasets with too many graphs for exact kernel methods such as PM.*

D. Which embedding and aggregation methods work best in RGM, particularly for inductive learning?

Given that RGM takes node embeddings as input, it can be seen as a two-step process consisting of learning node representations and aggregating them into a feature map for a graph. Here we consider alternative design choices per step.

Task. First, we compare three embedding approaches before constructing RGM feature maps: **node2vec** [9], **struc2vec** [36], and **xNetMF** [13]. These choices reflect different network embedding objectives [37]: node2vec preserves proximity between nodes, whereas the latter two preserve structural similarity. Moreover, xNetMF is designed for multi-network settings, whereas the other two are designed for single-network settings.

We embed all graphs in training folds together, followed by embedding all test graphs in a separate step (i.e., **inductive learning**). To embed graphs jointly using the single-network formulation of node2vec ands struc2vec, we combine their adjacency matrices as blocks as a single block-diagonal adjacency matrix. We perform 10 random walks of length 80, use a window size of 10, and set the embedding dimensionality to $d = 100$. For node2vec we set $p = q = 1$.

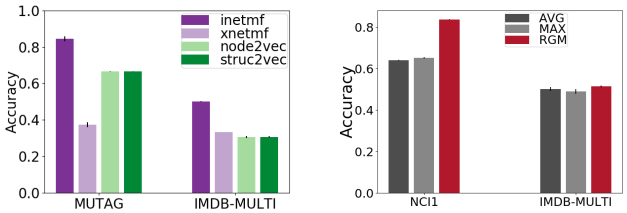
Results. We see in Figure 5a that off-the-shelf node2vec, struc2vec, and xNetMF all perform poorly as base node embedding methods for RGM. node2vec and struc2vec are designed for a single-graph setting, and even xNetMF, although designed for cross-network tasks, assumes a *transductive* setting where all graphs are given up front. In all cases, the feature space learned for the training graphs is *not guaranteed to be comparable to that learned for the test graphs*. However, our modification of xNetMF, iNetMF, performs dramatically better than its transductive counterpart, as well as node2vec and struc2vec. It succeeds in embedding nodes in test data into the subspace spanned by the training landmarks.

Observation 4. RGM *can successfully use advances in node embedding to classify graphs. The most important change that existing embedding methods may need, however, is a way to ensure continuity of the latent feature space* across training and test networks.

For iNetMF, there is little difference in performance compared to a transductive setting. This would also be true of RPF and EIG embeddings, where the computation can be done separately for each graph. However, most work in node representation learning optimizes an objective to preserve relative similarities between nodes [7]. Without care, such methods may be led astray in an inductive setting.

Next, we consider alternatives for aggregating embeddings.

Task. We compare our RGM feature maps using iNetMF embeddings to two alternative graph representations using feature pooling, which we call **AVG** and **MAX**. These create a d-dimensional feature vector by taking the average or maximum value, respectively, along each feature dimension.



(a) Node embedding methods for inductive classification **(b) RGM vs. embedding pooling methods**

Fig. 5: Best choices for node embedding and aggregation. An inductive graph classification setting shows that node embedding methods designed to preserve relative similarities between nodes that are being jointly embedded (e.g. at training or test time) may lead to incomparability between training and test graphs' embeddings. Given suitable node embeddings, RGM works better than simple pooling methods, which less fully capture the distribution of node embeddings.

Results. We see that in Figure 5, in terms of constructing feature representations of graphs, the pooling operations MAX and AVG yield inferior performance to our RGM variants. The margin is larger on graphs with node labels, as we illustrate with the largest labeled graph NCI1; it is smaller on the unlabeled collaboration network IMDB-MULTI. These results confirm the benefits of capturing the embedding distribution more comprehensively with RGM.

Observation 5. *Capturing the full distribution of embeddings using* RGM *is more expressive than pooling the embeddings using simple summary statistics such as mean or max.*

V. Related Work

In this section we outline related literature in three directions; see [42] for an overview of network similarity methods from a practitioner's perspective. Table IV qualitatively compares RGM to selected baselines with respect to our three desiderata: (**P1**) expressive and inductive graph comparison; (**P2**) efficient comparison; (**P3**) downstream use of fast machine learning models for graph classification.

Graph kernels. Some graph kernels capture graph similarity from substructures, such as walks [46], shortest paths [5], subtrees [27], graphlets [38], or other subgraphs [21]. Others leverage dependencies between these substructures [50], study propagation patterns [30], or characterize a restricted, strictly transductive class of valid optimal assignment kernels [25]. Finally, recent work has considered the tradeoffs between using explicit features and the implicit feature mappings of a kernel function [24], also for a restricted class of graph kernels.

Some works do consider node embeddings for graph classification: [18] considers optimal assignment of geometric embeddings, but produces indefinite similarity matrices. RetGK graph kernels [53] compute return probabilities of random walks in cubic time. The faster of the two proposed methods, RetGK_{II}, simply averages node feature maps and still applies the kernel trick at the end. More relevant to our work is [32], which apply the PM kernel [8] to embeddings formed from the

top eigenvectors of a graph's adjacency matrix. We achieve a similar design in a flexible *explicit feature map* that allows for faster training. Finally, RGM compares largely favorably to the concurrently proposed RGE random feature map [48] that approximates an EMD-like transportation distance between eigenvector embeddings. RGE samples node embeddings without discerning how they are distributed in vector space, the very information that RGM captures.

Techniques inspired by the Weisfeiler-Lehman test of isomorphism [40] can improve the performance of methods that use node labels, including ours. Further extensions capture global and local structure, although they require approximation to be computationally practical [28]. In general, computing graph kernel functions and using them in kernel machines falls short on computational properties laid out in **P2** and **P3**.

Other graph similarity functions (besides kernels) include graph edit distance [4], whose computational impracticality for all but small graphs violates **P2**. The scalable graph similarity function DeltaCon [23] is designed for graphs defined over the same set of vertices, limiting its expressivity (**P1**).

Unsupervised feature mappings. An early graph feature map, NetSimile, consists of basic summary statistics from distributions of hand-engineered node and edge features. Such features may be useful for aligning graphs [12] or exploratory graph analysis with domain knowledge [15] but are limited in expressivity. More recently, FGSD [45], uses histograms to characterize a graph based on its biharmonic kernel. However, its practical limitations include quadratic time complexity and inability to use node labels. NetLSD [44] was shown to be more powerful and scalable, but it too cannot use node label information. These all fall short on **P1** at minimum.

Like our method RGM, all of the above works are unsupervised, which makes training simpler and generally faster. Representations for graphs or subgraphs [1], [14] may also be learned by analogy to paragraph or document representation learning in NLP [26]. However, these methods require excessive amounts of graph sampling (a computational challenge for **P2**) to achieve competitive results and/or have high variance.

Deep neural networks. Deep neural networks have grown in popularity and have been extended to graph classification tasks. Diffusion-convolution neural networks [2] adapt graphs for use with existing convolutional architectures by scanning a diffusion process across each node, which had empirical limits for graph classification. PATCHY-SAN [31] extracts fixed-sizes patches from graphs and then uses graph canonization tools to define a vertex ordering for use with CNNs, which the recent work DGCNN [52] does in an end-to-end fashion.

It is also possible to adapt node classification architectures with specialized graph convolutions, such as GraphSAGE [10] and GCN [20], by aggregating the node features. We showed that given the same set of node embeddings, RGM aggregation is often more effective than the mean- and max-pooling operations that are often used in neural network architectures. The recent hierarchical method DiffPool [51] performs supervised node pooling at greater computational expense.

TABLE IV: Qualitative comparison of various methods. Existing graph kernels and unsupervised feature representations lack one or more desirable properties that RGM has.

Method	Expressive	Inductive	Fast Comparison	Fast ML
NetLSD	✗	✓	✓	✓
WLOA	✓	✗	✓	✗
RETGK	✓	✓	✗	✗
WLPM	✓	✓	✓	✗
RGM	✓	✓	✓	✓

It is challenging to make precise statements regarding **P1**, **P2**, and **P3** for deep learning-based methods, as all three depend on how well the training converges. In general, however, neural networks are heavily parametrized and thus more difficult to train, requiring additional computational resources such as GPUs (a practical efficiency issue regarding **P2** and **P3**) and risking overfitting especially on smaller datasets (a concern for expressivity, i.e. **P1**) [52]. Only recently have neural network models been designed for limited, noisy data in specific domains such as neuroscience [49].

Network embedding. Network representation learning has recently gained traction for its power in downstream graph mining tasks [7]. Intuitively, such methods learn similar embeddings for similar nodes. In most cases, similarity is defined in terms of proximity via random walks [34], [9] or first- and second-order connections [43]. Representations may be learned with shallow [34] or deep architectures [47], regardless of architecture, preserving within-graph node proximity may not be useful for multi-network problems [11]. Recent work inductively learns representations with deep convolutional architectures [10], but mainly targets tasks with node-level supervision. Node embedding that preserve *structural* node similarity may be more suitable for cross-network analysis [13]. struc2vec [36] samples context from an auxiliary structural similarity graph and optimizes a skip-gram objective to embed the nodes. However, struc2vec is still formulated for a *transductive* setting on a single graph. xNetMF [13] was designed for cross-network comparison, but still assumes a transductive multi-network setting. In contrast, *inductive* settings require embedding nodes in out-of-sample graphs.

VI. Conclusion

In this paper we propose RGM, a feature map that captures the distribution of a graph's node embeddings at multiple levels of resolution. We demonstrate theoretical connections between RGM and existing kernel methods, enhancing its performance with node labels using Weisfeiler-Lehman label expansion. We show that RGM is up to 20% more accurate than competitive baselines from graph kernels, feature learning, and deep neural networks. Furthermore, RGM is up to an order of magnitude faster and scales to larger datasets than the most relevant and competitive exact kernel baseline. RGM thus efficiently turns feature descriptors of nodes into a principled and powerful feature descriptor for the network.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. IIS 1845491, Army Young Investigator Award No. W911NF1810397, an Adobe Digital Experience award, an Amazon research faculty award, and a National Science Foundation Graduate Research Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or other funding parties.

REFERENCES

[1] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *PAKDD*. Springer, 2018.

[2] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, 2016.

[3] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network similarity via multiple social theories. In *ASONAM*. IEEE/ACM, 2013.

[4] David B Blumenthal and Johann Gamper. On the exact computation of the graph edit distance. *Pattern Recognition Letters*, 2018.

[5] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *ICDM*. IEEE, 2005.

[6] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *JMLR*, 6:2153–2175, 2005.

[7] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[8] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Efficient learning with sets of features. *JMLR*, 8:725–760, 2007.

[9] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 2016.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[11] Mark Heimann and Danai Koutra. On generalizing neural node embedding methods to multi-network problems. In *KDD MLG Workshop*, 2017.

[12] Mark Heimann, Wei Lee, Shengjie Pan, Kuan-Yu Chen, and Danai Koutra. Hashalign: Hash-based alignment of multiple graphs. In *PAKDD*. Springer, 2018.

[13] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. Regal: Representation learning-based graph alignment. In *CIKM*. ACM, 2018.

[14] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *ICML*, 2018.

[15] Di Jin and Danai Koutra. Exploratory analysis of graph data by leveraging domain knowledge. In *ICDM*. IEEE, 2017.

[16] Di Jin, Ryan A Rossi, Eunyee Koh, Sungchul Kim, Anup Rao, and Danai Koutra. Latent network summarization: Bridging network embedding and summarization. In *KDD*. ACM, 2019.

[17] Thorsten Joachims. Training linear SVMs in linear time. In *KDD*. ACM, 2006.

[18] Fredrik D Johansson and Devdatt Dubhashi. Learning with similarity functions on graphs using matchings of geometric embeddings. In *KDD*. ACM, 2015.

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

[20] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[21] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *NeurIPS*, 2016.

[22] Danai Koutra and Christos Faloutsos. *Individual and Collective Graph Mining: Principles, Algorithms, and Applications*. Synthesis Lectures on Dat Min and Knowl Disc. Morgan & Claypool Publishers, 2017.

[23] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*. SIAM, 2013.

[24] Nils Kriege, Marion Neumann, Kristian Kersting, and Petra Mutzel. Explicit versus implicit graph feature maps: A computational phase transition for walk kernels. In *ICDM*. IEEE, 2014.

[25] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *NeurIPS*, 2016.

[26] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[27] Pierre Mahé and Jean-Philippe Vert. Graph kernels based on tree patterns for molecules. *Mach. Learn.*, 75(1):3–35, April 2009.

[28] Christopher Morris, Kristian Kersting, and Petra Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *ICDM*. IEEE, 2017.

[29] Christopher Morris, Nils M Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *ICDM*. IEEE, 2016.

[30] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. Propagation kernels: efficient graph kernels from propagated information. *Mach. Learn.*, 102(2):209–245, 2016.

[31] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

[32] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI*, 2017.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, and et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*. ACM, 2014.

[35] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS*, 2008.

[36] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*. ACM, 2017.

[37] Ryan A Rossi, Di Jin, Sungchul Kim, Nesreen K Ahmed, Danai Koutra, and John Boaz Lee. From community to role-based graph embeddings. *arXiv preprint arXiv:1908.08572*, 2019.

[38] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 2009.

[39] Nino Shervashidze. Graph kernels in MATLAB. http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/, 2018.

[40] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12:2539–2561, November 2011.

[41] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *ALT*, 2007.

[42] Sucheta Soundarajan, Tina Eliassi-Rad, and Brian Gallagher. A guide to selecting a network similarity method. In *SDM*. SIAM, 2014.

[43] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*.

[44] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. In *KDD*. ACM, 2018.

[45] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *NeurIPS*, 2017.

[46] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Imre Kondor, and Karsten M. Borgwardt. Graph kernels. *JMLR*, 11:1201–1242, 2010.

[47] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*. ACM, 2016.

[48] Lingfei Wu, Ian En-Hsu Yen, Zhen Zhang, Kun Xu, Liang Zhao, Xi Peng, Yinglong Xia, and Charu Aggarwal. Scalable global alignment graph kernel using random features: From node embedding to graph embedding. In *KDD*. ACM, 2019.

[49] Yujun Yan, Jiong Zhu, Marlena Duda, Eric Solarz, Chandra Sripada, and Danai Koutra. Groupinn: Grouping-based interpretable neural network for classification of limited, noisy brain data. In *KDD*. ACM, 2019.

[50] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*. ACM, 2015.

[51] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[52] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[53] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *NeurIPS*, 2018.