A Fast Kernel for Attributed Graphs

Yu Su* Fangqiu Han* Richard E. Harang† Xifeng Yan*

Abstract

As a fundamental technique for graph analysis, graph kernels have been successfully applied to a wide range of problems. Unfortunately, the high computational complexity of existing graph kernels is limiting their further applications to larger-scale graph datasets. In this paper, we propose a fast graph kernel, the *descriptor matching* (DM) kernel, for graphs with both categorical and numerical attributes. The computation time of the DM kernel is linear with respect to graph size. On graphs with n nodes and m edges, the kernel computation for two graphs can be done in $O(n+m)$ time. Although there are other linear-time graph kernels, most of them are restricted to graphs with only categorical attributes; their efficiency mainly comes from the sparseness of the feature space resulted from the mutually orthogonal categorical attributes. Extensive experiments on both synthetic and real-world graph datasets show promising performance of DM in both accuracy and efficiency.

1 Introduction

Large graph databases are increasingly popular in many domains such as chemoinformatics [3], bioinformatics [5] and the web [2]. These graph datasets are characterized by their rich attribute information. For example, in chemoinformatics, molecules are often modeled as graphs, with atoms being nodes and covalent bonds being edges. Rich attributes are associated with both nodes and edges: categorical attributes on nodes like element types, numerical attributes on nodes like their partial charges and on edges like the spatial distance between elements. Many interesting questions arise with these graph datasets, e.g., how to predict the mutagenicity of a chemical compound by comparing its graph representation with other chemical compounds having known functionality?

Graph kernels have been successfully applied to various graph problems [5, 18]. A graph kernel is basically a function measuring the similarity of two graphs. A significant advantage of the kernel method is that it can decouple data representation from the learning machines: As long as a graph kernel is provided, readily-available learning machines like SVM [6] or the kernel PCA [20] become directly applicable.

The large scale and heterogeneous attributes of modern graph data call for graph kernels which are (1) efficient to compute and (2) capable of handling the rich attribute information on nodes and edges. More specifically, we argue that a *linear-time* graph kernel that can handle *both* categorical and numerical attributes is desired, while being linear-time means the runtime scales linearly with respect to the graph size $n + m$, where n is the number of nodes and m the number of edges. Few graph kernels proposed so far achieve the two goals simultaneously. Some graph kernels [25, 15, 8] achieve linear-time computation. However, they are restricted to graphs with only categorical attributes since their efficiency mainly comes from the sparseness of the feature space resulted from the mutually orthogonal categorical attributes. A few recent graph kernels [10] try to speed up computation on graphs with numerical attributes. Unfortunately, they are not linear-time kernels.

In this work, we propose a linear-time kernel for graphs with both categorical and numerical attributes. The proposed kernel, which we denote as the descriptor matching (DM) kernel, is based on a simple idea: Map each graph into a set of vectors (descriptors), and then apply a set-of-vectors matching kernel to measure graph similarity. We first propose a propagation based algorithm to generate feature vectors on nodes in linear time. By propagating categorical attributes along edges, we are able to generate a real vector for each node which encodes its attributes as well as its neighborhood information. Two nodes with similar attributes and neighbors will have similar vector representations; computing the similarity of two graphs therefore resorts to matching the vectors of their nodes. We then adapt the well-known Vocabulary-Guided pyramid matching (VG) kernel [13] to identify an approximately optimal matching between two vector sets, which is also done in linear time. We rigorously prove the linear scalability of the DM kernel. The most related work is the propagation kernel [22], which also propagate attribute information. It is initially proposed as a linear-time kernel for graphs with categorical attributes, and is recently extended to handle numerical attributes [21]. We will discuss about the differences later in the paper and also empirically compare with it.

We empirically experiment on both synthetic datasets and real-world datasets from chemo- and bioinformatics, and compare DM with several state-of-the-

*University of California, Santa Barbara. {ysu, fhan, xyan}@cs.ucsb.edu.

†U.S. Army Research Lab. richard.e.harang.ctr@mail.mil.

art graph kernels. Experiments on synthetic datasets confirm the linear scalability of the DM kernel. On real-world datasets, DM shows competitive performance in both classification accuracy and efficiency. Particularly, the experiment results demonstrate that DM can well exploit additional numerical attributes to improve classification accuracy, as opposed to when only using categorical attributes. Another salient characteristic of DM shown by the experiments is that its classification performance is very stable across different tasks. Even when its accuracy is not the best on a dataset, the difference to the best is usually small.

2 Related Work

We summarize existing graph kernels with an emphasis on computational complexity. The computation of a graph kernel is often done in two steps: (1) decomposing each graph into a set of features, and (2) comparing feature sets. In order to achieve overall linear scalability, a graph kernel has to be linearly scalable in both steps.

For the first step, many graph kernels choose to exhaustively enumerate a certain type of features in a graph, such as random walks [12, 16], paths [1], shortest paths [4, 10], subtrees [19], etc. Although algorithms have been proposed to reduce the effect of combinatorial explosion, due to their exhaustive nature, these kernels are still inefficient and hard to be applied to large graphs with hundreds or more nodes. A few recently proposed graph kernels achieve linear scalability in the first step by limiting the size of their feature space [25, 15, 8]. Our kernel follows the same strategy: A graph is decomposed into a set of vectors on nodes. The idea of propagating categorical attributes to get local feature vectors is also employed by some other kernels [22, 27, 21], where a random walk based propagation scheme is used. However, as we show in Appendix A, the random walk based propagation process, if run for enough iterations, will end up with feature vectors irrelevant to the initial labeling of the nodes and their neighbors. Our propagation scheme, as we will present soon, generate feature vectors well encoding the labeling and neighborhood information.

The linear scalability in the second step is harder to achieve. Comparing all possible feature pairs in two sets results in a quadratic time complexity. Linear-time comparison becomes possible when graphs have only categorical attributes, which yields a sparse discrete feature space [25, 15, 8]. Take [25] for example. It decomposes a graph into a set of size-limited subtrees, hashes each subtree into a string, and then counts common strings via string equality check. However, for graphs with numerical features this strategy fails, as we have to take the similarity of the continuous features into account, other than merely making a binary decision of whether two features are the same. [22, 21] try to tackle this problem via locality sensitive hashing, which is basically putting feature vectors into some uniform bins and then count. We employ a different approach. From a geometric point of view, our method identifies where the feature vectors really reside in the feature space and divide the space into non-uniform bins based on the real data distribution.

Our kernel seeks for a one-to-one matching between two sets of features, for which an efficient solution exists. The graph kernels proposed in [11] and [27] try to find an *optimal* one-to-one matching for their specific type of features. Unfortunately, they are not efficient and are not positive semi-definite kernels [29]. Our kernel efficiently identifies an approximately optimal correspondence between two feature sets by employing an existing set-of-vectors matching kernel, the VG kernel [13], whose computational complexity is linear with respect to the set size with mild adaptation.

3 Preliminaries

Following convention, we define an undirected graph G as a 4-tuple $(V, E, \mathcal{L}_c, \mathcal{L}_n)$, where V is the set of nodes, E the set of edges, and $\mathcal{L}_c$ and $\mathcal{L}_n$ the labeling function for categorical and numerical attributes, respectively. $\mathcal{L}_c : V \to \Sigma$, where $\Sigma = l_1, \ldots, l_L$ is the alphabet of categorical attributes. The labeling function for numerical attributes $\mathcal{L}_n : V \to \mathbb{R}^K$ assigns K numerical attributes to each node. For simplicity, we will work on a graph dataset with N graphs, and each graph has n nodes and m edges. We define graph size as $n + m$, and call a graph kernel a *linear-time* kernel if its runtime complexity is linear to graph size. $\mathcal{N}(v)$ is the neighborhood of node v, which is the set of nodes directly connected to v.

Throughout the paper, we will use the term *set* to denote a multiset which allows duplicate elements. Given two sets $\mathbf{X}$ and $\mathbf{Y}$ where $n_1 = |\mathbf{X}|$, $n_2 = |\mathbf{Y}|$, and $n_1 \leq n_2$, a one-to-one correspondence or a matching $\mathcal{M}(\mathbf{X}, \mathbf{Y}; \pi) = \{(\mathbf{x}_i, \mathbf{y}_{\pi_i}) | 1 \leq i \leq n_1\}$ matches every element in $\mathbf{X}$ to some unique element in $\mathbf{Y}$. $\pi = [\pi_1, \ldots, \pi_{n_1}], 1 \leq \pi_i \leq n_2$ is a permutation of indices where π_i specifies a match $(\mathbf{x}_i, \mathbf{y}_{\pi_i})$, for $1 \leq i \leq n_1$. We will use the terms one-to-one correspondence and matching interchangeably in the rest of the paper. We will also use the terms attribute and label interchangeably. We follow the kernel foundation in [24].

Definition 3.1. (Gram Matrix) *Let $\mathcal{X}$ be a nonempty set. Given a function $k : \mathcal{X}^2 \to \mathbb{R}$ and elements $x_1, \ldots, x_m \in \mathcal{X}$, the $m \times m$ matrix K with elements $K_{ij} \coloneqq k(x_i, x_j)$ is called the* gram matrix *(or*

kernel matrix*) of k with respect to $x_1, \ldots, x_m$. A gram matrix is p.s.d. if it is a positive semi-definite matrix.*

DEFINITION 3.2. ((VALID) KERNEL) *Let $\mathcal{X}$ be a nonempty set. A function k on $\mathcal{X} \times \mathcal{X}$ which for all $m \in \mathbb{N}$ and all $x_1, \ldots, x_m \in \mathcal{X}$ gives rise to a p.s.d. gram matrix is called a* valid kernel*, or a p.s.d. kernel. We will simply refer to it as a* kernel.

It is easy to construct new kernels from existing ones. Given two kernels k_1 and k_2, and $\alpha_1, \alpha_2 \geq 0$, $\alpha_1 k_1 + \alpha_2 k_2$ is still a kernel, and the pointwise product $k_1 k_2$ defined as $(k_1 k_2)(x_1, x_2) \coloneqq k_1(x_1, x_2) k_2(x_1, x_2)$ is also a kernel [24].

4 Descriptor Matching Kernel

4.1 Local Descriptor. We first introduce a concept, *local descriptor*. A local descriptor (or simply descriptor) is a fixed-length real-valued vector associated with a node. It encodes the labeling information of the node, as well as the topological and labeling information in its neighborhood, thus serving as the *identity* of the node: *Similar nodes should have similar descriptors.* Descriptor similarity is defined based on their Euclidean distance, while node similarity is defined in a recursive manner: Two nodes are more similar if their attributes and neighborhood are more similar. With this property, it becomes meaningful to measure graph similarity by matching their node descriptors. A *descriptor generator* f is a function mapping a node v to a descriptor $f(v) \in \mathbb{R}^D$, where $D = \|f(v)\|$. $\mathcal{F}(G) = \{f(v) | v \in V(G)\}$ is the descriptor set of a graph G.

Now we define our descriptor generator. The basic idea is to capture the labeling *and* neighborhood information about a node by propagating categorical attributes. The outcome of the propagation process is a series of feature vectors for each node. The continuous features are the key for incorporating numerical attributes. Since the features are continuous, numerical attributes can be directly appended to the feature vectors. The idea is that the numerical attributes of a node, such as the partial charge value of an atom in a molecule, are a direct part of the node's identity. Other linear-time graph kernels like [25, 15, 8] are hard to incorporate numerical attributes because their features are discrete.

For better presentation, we first leave out numerical attributes. Because of the recursive nature of the node similarity definition, it is natural to generate descriptors via an iterative process in which nodes exchange information with their neighborhood. We therefore define the Stochastic Cascade (SC) descriptor generator. The SC descriptor of a node v, $f_{sc}(v) = (A_1(v), \ldots, A_L(v))$, is a vector of length L, with the ith component $A_i(v)$ indicating the *strength of association* between the categorical attribute l_i and the node. Intuitively, the more nodes with attribute l_i there are in $\mathcal{N}(v)$, the stronger the association will be. Let $\eta \in [0, 1]$ be a scalar, h be the number of iterations, we model this intuition via the following iterative process, which generates a sequence of descriptors $f_{sc}^{(r)}(v) = (A_1^{(r)}(v), \ldots, A_L^{(r)}(v)), 0 \leq r \leq h$ for v:

(1) Initialization:

$$A_i^{(0)}(v) = \begin{cases} 1 & \text{if } \mathcal{L}_c(v) = l_i, \\ 0 & \text{otherwise;} \end{cases}$$

(2) Updating:

$$A_i^{(r+1)}(v) = \begin{cases} 1 & \text{if } A_i^{(r)}(v) = 1, \\ 1 - \prod_{u \in \mathcal{N}(v)} (1 - \eta A_i^{(r)}(u)) & \text{otherwise,} \end{cases}$$

$$\text{for } i = 1, \ldots, L,\ 0 \leq r < h.$$

To understand the above process, let's focus on the attribute l_1. In the beginning of iteration r, the strength of association $A_1^{(r)}(v)$ is regarded as the probability of v propagating l_1 to all of its neighbors. Here η is a decay factor, or can be thought as the loss ratio of propagation. Initially, $A_1^{(0)}(v)$ is set to 1 if l_1 is v's categorical attribute, and otherwise 0. In each iteration k, $A_1^{(r+1)}(v)$ is updated to the probability of the node receiving *at least one* l_1 from its neighborhood: the probability of the neighboring node u not propagating l_1 to v in iteration r is $1 - \eta A_1^{(r)}(u)$, so the probability of v not receiving any l_1 from $\mathcal{N}(v)$ is $\prod_{u \in \mathcal{N}(v)} (1 - \eta A_i^{(r)}(u))$, therefore we end up with the above updating rule.

A competitor of our SC descriptor generator is a descriptor generator based on a random walk on graphs, which, although termed differently, has been exploited in some way in [22, 27, 21]. But we prove in Appendix A that it does not have the descriptor property, i.e., similar nodes should have similar descriptors. Two nodes in a graph, as long as they have the same degree, will always end up with the same descriptors irrelevant to the initial labeling of the nodes and their neighborhoods[1].

THEOREM 4.1. *The SC descriptors for N graphs can be computed in time $O(NLhm)$.*

Proof. In each iteration, each categorical attribute in Σ will be propagated for at most $2m$ times, and each propagation will incur an $O(1)$ number of operations,

[1] [21] suggested that, pragmatically, random walk based propagation can stop early without getting into the stationary states. We apply this strategy in evaluation.

so the overall runtime complexity of computing SC descriptors for N graphs and h iterations is $O(NhmL)$.

Numerical attributes are directly appended to the descriptors defined above. We normalize each numerical attributes to $[0, 1]$.

4.2 Descriptor Matching Kernel.

Definition 4.1. (Descriptor Matching Kernel) *Given a base kernel k defined on sets of vectors, if we denote the set of SC descriptors of graph G in the rth iteration as $\mathcal{F}^{(r)}(G)$, the* descriptor matching kernel *k_{dm} on two graphs G_1 and G_2 is defined as:*

$$k_{dm}^{(h)}(G_1, G_2) = \sum_{r=0}^{h} k(\mathcal{F}^{(r)}(G_1), \mathcal{F}^{(r)}(G_2)).$$

Theorem 4.2. *For any $h \in \mathbb{N}$, $k_{dm}^{(h)}$ is positive semi-definite (p.s.d.) if k is p.s.d.*

Proof. This follows directly from the fact that p.s.d. kernels are closed under addition.

The next step is to find a base kernel defined for two sets of vectors. There are three requirements for the base kernel: (1) Its computation must be efficient. More specifically, its time complexity should be linear with respect to graph size. (2) It should measure the similarity of two sets of vectors in an intuitive manner. (3) It is able to handle high-dimensional vectors. Putting all these requirements together, we choose the VG kernel [13] from computer vision. It identifies a one-to-one correspondence between two sets of vectors via non-uniform quantification, which makes it suitable for high-dimensional vectors since it can locate where the vectors really reside in the high-dimensional space and divide the space accordingly. Although the original VG kernel did not claim to be linearly scalable, we show next that with mild modification, its time complexity becomes linear with respect to graph size.

4.3 VG Kernel. Given a descriptor generator f and two graphs G_1 and G_2, we now discuss how to define a kernel k to efficiently measure the similarity of their corresponding descriptor sets $\mathcal{F}(G_1)$ and $\mathcal{F}(G_2)$. Suppose $\mathcal{F}(G_1) = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_1}\}$, $\mathcal{F}(G_2) = \{\mathbf{y}_1, \ldots, \mathbf{y}_{n_2}\}$, $n_1 \leq n_2$ and $\mathcal{M}(\mathcal{F}(G_1), \mathcal{F}(G_2); \pi)$ is a matching from $\mathcal{F}(G_1)$ to $\mathcal{F}(G_2)$, a set-of-vectors matching kernel k is defined as follow:

$$k(\mathcal{F}(G_1), \mathcal{F}(G_2)) = \sum_{i=1}^{n_1} w(\|\mathbf{x}_i - \mathbf{y}_{\pi_i}\|_2),$$

where $w(\cdot)$ is a weighting function. Note that under this definition k is not necessarily p.s.d.

Now the problem boils down to finding an appropriate matching. The most intuitive way is to find the optimal matching that maximizes $k(\mathcal{F}(G_1), \mathcal{F}(G_2))$, which can be formulated as the classic maximum weighted bipartite matching problem and solved by prominent algorithms such as the Hungarian algorithm [11, 27]. However, it is not favorable for two reasons: (1) The computational complexity is rather high (cubic), and (2) it results in a kernel which is not p.s.d. [29]. Another solution is discretization [22]. The idea is to map a vector into a 1-d histogram, and efficiently match vectors based on whether they fall into the same bin. It scales linearly, but the main problems are: (1) Bins are unweighted, or in other word, w is a constant function; (2) bins are orthogonal, so vectors in different bins are never matched. Nevertheless, the linear computational complexity is appealing. We choose the Vocabulary-Guided (VG) pyramid matching kernel, which is based on a somewhat similar idea, but in a more sophisticated manner. It aims to efficiently find an *approximately optimal* matching, and elegantly solves both of the problems via replacing the 1-d histogram by a data-dependent multi-resolution histogram with non-uniformly shaped bins. We next reformulate it in a way suitable for our descriptor sets, and adapt it to ensure its linear scalability.

Pyramid construction. Suppose $\mathcal{G}$ is a set of N graphs and $\mathcal{F}(\mathcal{G}) = \{f(v) | v \in G, G \in \mathcal{G}\}$. The VG kernel starts off by partitioning the descriptor space into a pyramid of non-uniformly shaped regions/bins, which is built by performing hierarchical clustering on $\mathcal{F}(\mathcal{G})$. The pyramid structure is controlled by two hyper-parameters, the number of levels t, and the branching factor b. The jth bin at the ith level is denoted as $B_j^{(i)} = (\mathbf{X}_j^{(i)}, c_j^{(i)}, s_j^{(i)})$, where $c_j^{(i)}$ is its center, $s_j^{(i)}$ its diameter with $s_j^{(i)} = \max\{\|\mathbf{x_1} - \mathbf{x_2}\| \,|\, \mathbf{x_1}, \mathbf{x_2} \in \mathbf{X}_j^{(i)}\}$, and $\mathbf{X}_j^{(i)} \subseteq \mathcal{F}(\mathcal{G})$ the set of descriptors in the bin. Then the pyramid is denoted as $\{B_j^{(i)}\}_{0 \leq i \leq t-1, 1 \leq j \leq b^i}$, and is constructed as in Algorithm 1.

Lines 4, 9, and 10 compute bin diameters, i.e., the maximum distance between any two descriptors in the bin. The original VG kernel will compute the distance between each pair of descriptors and find the maximum, which results in a quadratic time complexity. We approximate it by two upper bounds, the doubled maximum distance from any descriptor in the bin to the center of the bin, and the diameter of the parent bin, as shown at line 9 and 10, respectively. This grants us linear scalability. Bin diameters are critical and will be used to compute bin weights. Later in §5 we empirically demonstrate that the DM kernel built

Algorithm 1 Pyramid construction

1: **Initialization**:
2: $\mathbf{X}_1^{(0)} \leftarrow \mathcal{F}(\mathcal{G})$
3: $c_1^{(0)} \leftarrow 2 \times \frac{1}{|\mathcal{F}(\mathcal{G})|} \sum_{\mathbf{x} \in \mathcal{F}(\mathcal{G})} \mathbf{x}$
4: $s_1^{(0)} \leftarrow \max_{\mathbf{x} \in \mathcal{F}(\mathcal{G})} \|\mathbf{x} - c_1^{(0)}\|$
5: **for** $i = 0$ **to** $t-2$ **do**
6: **for** $j = 1$ **to** b^i **do**
7: run k-means clustering to partition $B_j^{(i)}$ into b child bins $\{B_k^{(i+1)}\}_{(j-1)b+1 \le k \le jb}$
8: **for** $k = (j-1)b+1$ **to** jb **do**
9: $s_k^{(i+1)} \leftarrow 2 \times \max_{\mathbf{x} \in \mathbf{X}_k^{(i+1)}} \|\mathbf{x} - c_k^{(i+1)}\|$
10: $s_k^{(i+1)} \leftarrow \min(s_k^{(i+1)}, s_j^{(i)})$

on the approximated VG kernel achieves promising performance in both efficiency and accuracy.

Multi-resolution histogram construction. Given a graph G and its descriptor set $\mathcal{F}(G)$, a multi-resolution histogram is constructed according to the pyramid structure. The multi-resolution histogram is defined as $\Psi(G) = [H^{(0)}(G), \ldots, H^{(t-1)}(G)]$, where $H^{(i)}(G) = [H_1^{(i)}, \ldots, H_{b^i}^{(i)}]$ is a 1-d histogram with b^i bins at the ith level, $0 \le i \le t-1$. Algorithm 2 shows how to construct $\Psi(G)$ by walking each descriptor through the pyramid and identifying its bin memberships along the way, where $p = (p_0, \ldots, p_{t-1})$ is a vector with p_i being the index of the bin where the descriptor is located at level i, $0 \le i \le t-1, 1 \le p_i \le b^i$.

Algorithm 2 Multi-resolution histogram construction

1: **for** $\mathbf{x} \in \mathcal{F}(G)$ **do**
2: $p_0 \leftarrow 1$
3: $H_1^{(0)} \leftarrow H_1^{(0)} + 1$
4: **for** $i = 1$ **to** $t-1$ **do**
5: $p_i \leftarrow \text{argmin}_j \|c_j^{(i)} - \mathbf{x}\|, (p_{i-1}-1)b+1 \le j \le p_{i-1}b$
6: $H_{p_i}^{(i)} \leftarrow H_{p_i}^{(i)} + 1$

Matching multi-resolution histograms. The matching process goes from the finest level ($i = t-1$) to the coarsest level ($i = 0$). In this way, we will first consider matching the closest descriptors (at level $t-1$), and as we climb to the higher levels in the pyramid, increasingly further descriptors are allowed to be matched. Given two multi-resolution histograms $\Psi(G_1)$ and $\Psi(G_2)$, the number of matches found in $B_j^{(i)}$ is derived via bin intersection:

$$\mathcal{I}_j^{(i)} = min(H_j^{(i)}(G_1), H_j^{(i)}(G_2)).$$

The number of *new* matches found in a bin is computed by subtracting the number of matches found in all its child bins from $\mathcal{I}_j^{(i)}$, which is the *true* number of descriptors matched in this bin:

$$\mathcal{J}_j^{(i)} = \begin{cases} \mathcal{I}_j^{(i)}, & i = t-1; \\ \mathcal{I}_j^{(i)} - \sum_{k=(j-1)b+1}^{jb} \mathcal{I}_k^{(i+1)}, & 0 \le i \le t-2. \end{cases}$$

We give an example in Appendix B to illustrate the above process. The VG kernel is defined as follow, where $w_{ij} = \frac{1}{1+s_j^{(i)}}$ is the weight of $B_j^{(i)}$ measuring how much a match found in the bin contributes to the overall similarity:

DEFINITION 4.2. (VG KERNEL) *Given two descriptor sets $\mathcal{F}(G_1)$ and $\mathcal{F}(G_2)$, and the corresponding multi-resolution histograms $\Psi(G_1)$ and $\Psi(G_2)$, the* VG kernel *k_{vg} is defined as:*

$$k_{vg}(\mathcal{F}(G_1), \mathcal{F}(G_2)) = \sum_{i=0}^{t-1} \sum_{j=1}^{b^i} w_{ij} \mathcal{J}_j^{(i)}. \tag{4.1}$$

THEOREM 4.3. *k_{vg} is p.s.d.*

Proof. We re-write Eq. (4.1) as $k_{vg}(\mathcal{F}(G_1), \mathcal{F}(G_2)) = \sum_{i=0}^{t-1} \sum_{j=1}^{b^i} (w_{ij} - p_{ij}) \mathcal{I}_j^{(i)}$, where p_{ij} is the weight associated with the parent bin of $B_j^{(i)}$, and that for $B_1^{(0)}$ is set to 0. Since the bin intersection $\mathcal{I}$ is a p.s.d. kernel [23], and since p.s.d. kernels are closed under addition and scaling by a positive scalar, k_{vg} is a valid kernel as long as $w_{ij} >= p_{ij}$ for all bins. This is guaranteed by (1) w_{ij} is a monotonic decreasing function with respect to $s_j^{(i)}$, and (2) $s_j^{(i)}$ is not bigger than the diameter of its parent bin, which is guaranteed by the line 10 of Algorithm 1.

THEOREM 4.4. *Given N graphs and their corresponding descriptor sets, suppose the maximum number of iterations for k-means clustering is H, the N-by-N kernel matrix of k_{vg} can be computed in $O(N(Hb+N)tn)$.*

Proof. Let us examine the time complexity of each step.

First, the pyramid can be built in $O(HNntb)$. On one hand, the hierarchical clustering can be performed in $O(HNntb)$. It takes at most $O(Hb)$ operations to determine the bin membership for each descriptor at each level, and there are in total Nn descriptors. On the other hand, determining all of the bin diameters can be done in $O(Nnt)$, because at each level, each descriptor will be accessed exactly once. So the pyramid construction takes $O(HNntb)$ time.

Second, the N multi-resolution histograms can be constructed in $O(Nntb)$. It can be seen from that, for each of the n descriptor, it takes b comparisons to determine its bin membership at each level.

Finally, matching all pairs of multi-resolution histograms takes $O(N^2nt)$ time. Matching two multi-resolution histograms can be done in $O(nt)$ time via a sparse representation of the multi-resolution histograms which only stores non-empty entries, and there are N^2 pairs to match. For implementation details, see [13].

Therefore, the overall time complexity is $O(HNntb + Nntb + N^2nt) = O(N(Hb + N)tn)$.

Theorem 4.4 asserts the linear scalability of the VG kernel, which paves the way to the proof of the linear scalability of the DM kernel.

THEOREM 4.5. *With k_{vg} as the base kernel, k_{dm} on a pair of graphs can be computed in a linear time with respect to graph size.*

Proof. For N graphs, directly following Theorem 4.1 and Theorem 4.4, k_{dm} can be computed in $O(NLhm + N(Hb + N)htn) = O(Nh(Lm + Htbn) + N^2htn)$ time, where h is the number of iterations, and the amortized cost for a pair of graphs is $O(\frac{h}{N}(Lm + Htbn) + htn) = O(\frac{1}{N}Lhm + (\frac{1}{N}Hb + 1)htn)$, which is linear with respect to graph size.

5 Evaluation

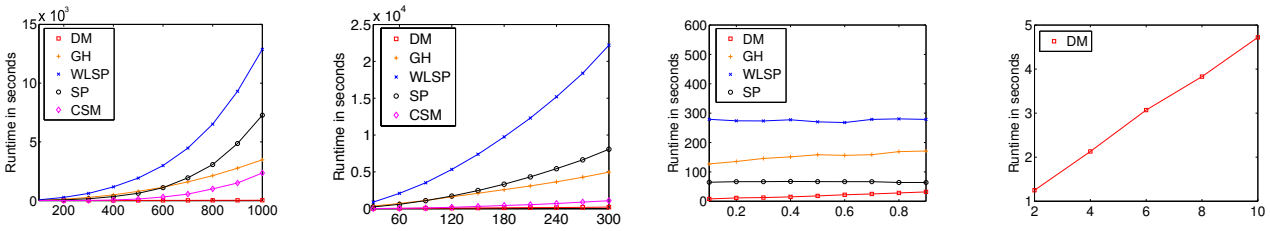
We compare DM with state-of-the-art graph kernels: the propagation kernel (PK) [22, 21], the Weisfeiler-Lehman subtree (WL) kernel [25], the Weisfeiler-Lehman shortest-path (WLSP) kernel [26], the shortest-path (SP) kernel [4], the connected subgraph matching (CSM) kernel [17], and the GraphHopper (GH) kernel [10]. DM, PK, WL, SP, WLSP and GH are implemented in Matlab, VG is implemented in C++, and CSM is implemented in Java. DM, PK and WL are linearly scalable while others are not. WL can only be applied on graphs with categorical attributes, while DM and PK can handle numerical attributes as well. In DM, we directly append numerical attributes to descriptors, while in PK, numerical attribtues are also propagated.

5.1 Runtime Analysis on Synthetic Datasets. In this experiment, we test graph kernels on randomly generated graphs with both categorical and numerical attributes. The main goal is to confirm the linear scalability of the DM kernel. The results of WL and PK are similar to DM and are omitted.

Experiment setup. We randomly generate undirected graphs based on two parameters: the number of graphs N, and the number of nodes n. The default values are $N = 10$ and $n = 200$. Average node degree is set to 5 so that graph size increases linearly with respect to n. n nodes are first generated, then edges are randomly inserted until a certain number is reached. We additionally experiment on graphs with varying density $\sigma = \frac{2m}{n(n-1)}$, and also evaluate DM with varying number of iterations h. For DM, the default value of h is 10, and t and b are set to 4 and 10, respectively. For WLSP, the number of iterations h is set to 3. For CSM, the maximum size of subgraphs k is set to 5. When evaluating one parameter, all other parameters are fixed to the default values. Node categorical and numerical attributes are randomly generated. The total CPU time to compute the N-by-N kernel matrix is reported.

Result analysis. The results are presented in Figure 1. Figure 1(a) shows the runtime behavior with respect to n. DM scales linearly with a small increase rate, while the runtime of other kernels increases at least quadratically. Figure 1(b) gives the runtime results with varying N. DM scales nearly linearly, which shows that the linear term with respect to N in the overall time complexity is dominating when N is moderate. In Figure 1(c), we show how the graph density, namely the number of edges m when n is fixed, affects the runtime of DM. As expected, the runtime of DM increases linearly. The result of CSM is not reported because its extremely high runtime when graphs are dense. We argue that, on real-world graphs, especially when graph size is large, m can rarely get up to $O(n^2)$, therefore a runtime complexity in $O(m + n)$ usually scales more elegantly than $O(n^2)$. Finally, Figure 1(d) shows that DM also scales linearly with respect to h.

5.2 Classification Performance on Real-world Datasets. We experiment with 11 well-accepted benchmark datasets from chemo- and bioinformatics. MUTAG [9] is a set of 188 chemical compounds labeled according to whether or not they have a mutagenic effect on a bacterium. ENZYMES [5] comprises of 600 enzymes represented by their secondary structure elements (SSEs), and the task is to classify each enzyme into one of the 6 EC top level classes. D&D is a datasets consisting of 1178 proteins where amino acids are modeled as nodes. The graphs are therefore much larger. The task is to predict whether a protein is an enzyme. The PTC [14] dataset contains chemical compounds labeled according to their carcinogenicity to rodents. Four datasets, mice (MM), female mice (FM), male rats (MR), and female rats (FR), are developed according to their effect on different rodents. We acquired the dataset from ChemDB [7]. We obtained four more chemical compound datasets from [28]: COX-2 is a dataset of 467 cyclooxygenase-2 inhibitors, BZR a dataset of 405 ligands for the benzodiazepine receptor, DHFR a dataset of 756 inhibitors of dihydrofolate reductase, and ER a dataset of 1,009 estrogen receptors. The task is to predict a chemical compound as active or



(a) Number of nodes n (b) Number of graphs N (c) Graph density σ (d) Number of iterations h

Figure 1: Runtime behavior on synthetic datasets.

Table 1: Statistics of the benchmark datasets

Dataset	MUTAG	ENZYMES	D&D	FR	FM	MR	MM	COX-2	BZR	DHFR	ER
# graphs	188	600	1178	344	351	336	349	303	306	393	446
# positive	125	-	691	121	143	152	129	148	157	126	181
# categorical attributes	8	3	82	20	19	19	21	7	8	7	10
Avg. # nodes	26.03	32.63	284	25.56	26.08	25.05	25.25	41.56	35.04	41.58	41.96
Max. # nodes	28	126	5748	109	109	109	109	56	57	71	93
Avg. # edges	27.89	62.14	716	25.96	26.53	25.4	25.62	43.8	37.5	43.71	43.96

inactive in a certain reaction.

All datasets have categorical attributes on nodes. MUTAG, MM, FM, MR and FR have a node numerical attribute, the partial charge of atoms. COX-2, BZR, DHFR and ER come with the 3D coordinates of atoms, based on which we compute the spatial distance between atoms, and use it as a numerical attribute on edges. We choose the 3D-length of the SSEs as a node numerical attribute for ENZYMES. The dataset statistics are reported in Table 1.

Evaluation scheme. We perform 10-fold nested cross-validation of C-Support Vector Machine provided by LIBSVM [6]. In each fold, all hyper-parameters are optimized by an extra 9-fold cross-validation on the training data only. The whole process is repeated for 10 times, and the mean and standard deviation of the classification accuracy over the 10 runs are reported. The reported runtime is obtained by running each kernel with the hyper-parameters most frequently selected by the model selection process. The initialization time for each kernel is included. The "one-against-one" strategy is adopted for the multi-class classification on ENZYMES. See Appendix C for the detailed configuration.

Graphs with only categorical attributes. We first experiment on graphs with only categorical attributes. The results are shown in Table 3. A method is bold-faced in the table if it achieves the highest accuracy, or is not significantly worse than the highest according to the student t test at $p = 0.05$. The results show that our DM kernel achieves comparable accuracy with other kernels. It is in top 3 on all the datasets except COX-2, and achieves the highest accuracy on MUTAG and D&D.

In terms of efficiency, among the linear-time kernels, DM is comparable with WL while in general slower than PK. For the other kernels, GH and CSM are less efficient than DM. Because WLSP and SP utilize the hash-and-check-equality strategy (cf. §2), they are quite efficient on datasets with small graphs like MUTAG, COX-2, BZR, DHFR, and ER. However, these non-linear-time kernels are hard to scale to larger graphs, such as those in D&D. As a result, WLSP and CSM cannot finish within 2 days on D&D, SP takes over 4 hours, and GH takes 3 days. The reason why DM takes more time on D&D than WL is that the runtime of DM grows linearly with respect to L, the number of categorical attributes, while the time complexity of WL is not dependent on L, and $L = 82$ on D&D. Nevertheless, we can safely draw the conclusion that DM can scale to large graphs with a moderate number of categorical attributes, which is the common case in many applications.

Graphs with numerical attributes. We now test on graphs with additional numerical attributes, which are the main targets of this work. WL is not applicable in this case. Table 3 shows the experiment results. In terms of classification accuracy, DM is among the best on 9 out of the 10 datasets, and achieves the highest accuracy on 6 of them. The only kernel which is comparable in terms of overall classification performance is WLSP. PK, SP and GH are in general less competitive, while CSM can compete on several datasets. In terms of efficiency, since the incorporation of numerical attributes fails the hash-and-equality-check strategy, SP and WLSP become much slower. We

Table 2: Experiment results on graphs with only categorical attributes.

Method		MUTAG	ENZYMES	D&D	COX-2	BZR	DHFR	ER
DM	accuracy	**87.89±1.88**	59.48±0.89	**79.69±0.64**	73.97±1.80	75.80±1.10	**80.54±0.94**	**83.61±1.17**
	runtime	4"	27"	1h10'	29"	31"	27"	1'2"
PK	accuracy	84.22±1.47	46.43±1.26	**79.27±0.33**	75.33±2.34	**76.60±1.77**	**80.51±1.66**	81.91±0.78
	runtime	0.2"	2.9"	6'2"	1.5"	0.6"	4.2"	0.5"
WL	accuracy	**86.61±1.40**	53.22±1.30	79.01±0.43	**76.13±1.74**	**78.17±1.60**	**81.03±0.82**	82.52±0.86
	runtime	7"	28"	8'47"	17"	15"	22"	1'10"
SP	accuracy	85.94±1.94	43.20±1.21	78.26±0.76	73.97±2.33	72.83±1.87	75.18±0.97	76.93±1.22
	runtime	0.4"	3"	4h27'	1.4"	1"	2"	2"
GH	accuracy	82.89±1.69	37.98±1.57	75.80±0.46	71.90±2.15	72.93±1.46	74.00±1.40	78.36±1.02
	runtime	37"	12'11"	3d20h	4'24"	3'33"	6'50"	9'
WLSP	accuracy	85.72±1.96	**60.92±0.90**	-	72.47±1.35	**77.17±1.51**	78.95±1.29	**83.80±0.91**
	runtime	3"	1'26"	> 2 days	8"	7"	12"	14"
CSM	accuracy	85.61±1.95	58.68±1.03	-	**77.27±0.68**	71.43±1.91	78.87±0.82	78.80±0.92
	runtime	6'5"	8h24'	> 2 days	5'54"	22'45"	24'31"	4h9'

Table 3: Experiment results on graphs with numerical attributes.

Method		MUTAG	ENZYMES	COX-2	BZR	DHFR
DM	accuracy	**90.09±1.87**	**70.37±1.57**	**76.17±2.01**	**78.83±1.31**	**80.92±0.94**
	runtime	11"	44"	19"	52"	32"
PK	accuracy	83.56±1.15	55.38±1.21	74.80±2.55	72.00±2.41	79.67±1.23
	runtime	0.2"	3.3"	0.6"	0.9"	3.4"
SP	accuracy	87.11±1.73	**70.90±0.83**	72.03±1.17	74.60±2.35	77.28±1.14
	runtime	2'25"	19'20"	6'25"	4'15"	8'40"
GH	accuracy	85.78±2.50	62.33±1.07	71.27±2.87	73.10±1.76	74.08±1.21
	runtime	29"	9'	4'44"	3'42"	7'14"
WLSP	accuracy	**89.06±1.98**	**71.38±0.36**	**74.87±2.74**	**77.70±1.84**	78.54±1.07
	runtime	7'30"	32'6"	13'	18'55"	43'
CSM	accuracy	**90.61±2.39**	68.91±0.92	**75.03±1.63**	74.37±2.20	**79.72±1.66**
	runtime	6'25"	1'45"	6'41"	19'16"	35'35"

Method		ER	FR	FM	MR	MM
DM	accuracy	**83.77±1.17**	**66.83±1.26**	**61.94±1.34**	**60.79±1.59**	65.09±1.74
	runtime	1'10"	5"	13"	5"	6"
PK	accuracy	78.57±0.93	65.49±1.54	**61.03±2.61**	58.71±2.10	**66.76±1.36**
	runtime	3.3"	0.5"	1.1"	0.7"	0.3"
SP	accuracy	80.89±1.08	64.66±1.21	59.85±1.32	**60.44±1.38**	65.24±1.04
	runtime	13'	5'18"	5'5"	5'	4'40"
GH	accuracy	78.48±0.84	63.57±1.70	59.68±1.71	58.62±1.29	60.27±1.54
	runtime	9'39"	2'52"	2'40"	2'42"	2'33"
WLSP	accuracy	**83.73±0.97**	**66.09±2.19**	**62.62±2.00**	**59.88±1.53**	**67.03±1.41**
	runtime	49'50"	14'20"	13'46"	13'38"	12'47"
CSM	accuracy	80.16±0.79	**66.49±1.49**	60.71±1.77	58.24±2.37	**65.94±2.45**
	runtime	41'	1'53"	2'50"	13'57"	14'20"

compute the average ratio between the runtime of each method and the runtime of DM over all the datasets. DM is 29 times faster than SP, 18 times faster than GH, 80 times faster than WLSP, and 53 times faster than CSM. The other linear-time kernel, PK, is more efficient than DM because of its simplicity. However, it is less competitive in accuracy, being among the best in only two datasets. If we consider all the 17 datasets in both Table 2 and Table 3, DM is significantly better than PK on 11 datasets, while PK wins on 2 datasets (under student t test at $p = 0.05$).

Comparing the results on the 6 common datasets (MUTAG, ENZYMES, COX-2, BZR, DBFR, and ER) in Table 2 and Table 3 shows each kernel's capability of exploiting numerical attributes. Particularly, we see that the accuracy of DM increases on each dataset after incorporating numerical attributes. On the contrary, on 5 of the 6 datasets, the accuracy of PK actually decreases[2]. This may imply that directly appending the numerical attributes to the SC descriptors is an effective way of exploiting numerical attributes. A more in-depth analysis is of interest for future study.

Another salient characteristic of DM regarding accuracy is that, while the accuracy of other kernels vary

[2]On COX-2, BZR, DHFR and ER, we run two experiments for PK: One uses the the inverse of the 3D-distance as edge weight; the other uses the 3D coordinates as node attributes. PK performed much worse in the former. We report the latter.

from dataset to dataset, DM consistently gives good accuracy. Even on COX-2 and BZR in Table 2 where DM seems to fall short, the difference between the accuracy of DM and the highest is small (3.3% at most).

6 Conclusions

We introduced a linear-time graph kernel which can handle graphs with both categorical and numerical attributes. From experiments on both synthetic and real-world datasets, the proposed kernel showed promising performance in accuracy and efficiency. The proposed kernel is a good alternative to existing kernels for tasks involving small graphs. Moreover, it is among the first kernels applicable to large graphs with rich attributes.

References

[1] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC bioinformatics*, 9(Suppl 11):S2, 2008.

[2] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - the story so far, 2009.

[3] D. D. Bonchev and D. H. Rouvray. *Chemical graph theory: introduction and fundamentals*, volume 1. 1991.

[4] K. Borgwardt and H. Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.

[5] K. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl 1):i47–i56, 2005.

[6] C. Chang and C. Lin. Libsvm: a library for support vector machines. *TIST*, 2(3):27, 2011.

[7] J. H. Chen, E. Linstead, S. J. Swamidass, D. Wang, and P. Baldi. ChemDB update – full-text search and virtual chemical space. *Bioinformatics*, 23(17):2348–2351, 2007.

[8] F. Costa and K. D. Grave. Fast neighborhood subgraph pairwise distance kernel. In *ICML*, 2010.

[9] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[10] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. In *NIPS*, 2013.

[11] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *ICML*, 2005.

[12] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages 129–143. 2003.

[13] K. Grauman and T. Darrell. Approximate correspondences in high dimensions. In *NIPS*, 2006.

[14] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[15] S. Hido and H. Kashima. A linear-time graph kernel. In *ICDM*, 2009.

[16] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, 2003.

[17] N. Kriege and P. Mutzel. Subgraph matching kernels for attributed graphs. In *ICML*, 2012.

[18] U. Lösch, S. Bloehdorn, and A. Rettinger. Graph kernels for rdf data. In *the Semantic Web: Research and Applications*, pages 134–148. Springer, 2012.

[19] P. Mahé and J. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35, 2009.

[20] S. Mika, B. Schölkopf, A. J. Smola, KR Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In *NIPS*, 1998.

[21] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, pages 1–37, 2015.

[22] M. Neumann, N. Patricia, R. Garnett, and K. Kersting. Efficient graph kernels by randomization. In *Machine Learning and Knowledge Discovery in Databases*, pages 378–393. 2012.

[23] F. Odone, A. Barla, and A. Verri. Building kernels from binary strings for image matching. *IEEE Trans. on Image Processing*, 14(2):169–180, 2005.

[24] B. Schölkopf and A. J. Smola. *Learning with kernels*. The MIT Press, 2002.

[25] N. Shervashidze and K. Borgwardt. Fast subtree kernels on graphs. In *NIPS*, 2009.

[26] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. Borgwardt. Weisfeiler-lehman graph kernels. *the Journal of Machine Learning Research*, 12:2539–2561, 2011.

[27] A. Smalter, J. Huan, Y. Jia, and G. Lushington. GPD: a graph pattern diffusion kernel for accurate graph classification with applications in cheminformatics. *TCBB*, 7(2):197–207, 2010.

[28] J. J. Sutherland, Lee A. O'Brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *Journal of Chemical Information and Computer Sciences*, 43(6):1906–1915, 2003.

[29] J. Vert. The optimal assignment kernel is not positive definite. *arXiv preprint arXiv:0801.4061*, 2008.