

High Order Stochastic Graphlet Embedding for Graph-Based Pattern Recognition

Anjan Dutta and Hichem Sahbi

Abstract—Graph-based methods are known to be successful for pattern description and comparison purpose. However, a lot of mathematical tools are unavailable in graph domain, thus restricting the generic graph-based techniques to be applicable within the machine learning framework. A way to tackle this problem is graph embedding into high dimensional space in either an explicit or implicit manner. In this paper, we propose high order stochastic graphlet embedding (SGE) that explicitly embed a graph into a real vector space. Our main contribution includes a new stochastic search procedure that allows one to efficiently parse a given graph and extract or sample unlimitedly high order graphlets. We consider these graphlets with increasing size in order to model local features, as well as, their complex interactions. We also introduce or design graph hash functions with very low probability of collision to hash those sampled graphlets for partitioning them into sets of isomorphic ones and measure their distribution in large graph collections, which results in accurate graph descriptions. When combined with support vector machines, these high order graphlet-based descriptions have positive impact on the performance of graph-based pattern comparison and classification as corroborated through experiments on different standard benchmark databases.

Index Terms—Stochastic graphlets, Graph embedding, Graph classification, Graph hashing, Betweenness centrality.



1 INTRODUCTION

In this paper, we consider graph-based pattern recognition problem: given a pattern (image, shape, handwritten character, document etc.) represented as a graph, the goal is to predict the class that best describes the visual and semantic content of the pattern, which essentially turns into a *graph classification or recognition* problem. Although, here we focus on general graph categorization, while mentioning specific example, we mostly consider different cases from computer vision domain. This is mainly because of our domain specific interests and applications we are focused on, but our discussions and proposed solution are generalized enough and are applicable to other domains. In any case, most of the early pattern classification methods were designed upon numerical feature vectors resulted from some statistical analysis of pattern [10], [25]. Extensions of this model that integrate some kind of structural information most certainly surpass other methods [23]. It is indeed true that different parts of certain pattern do not appear independently and structural relationships among these parts are crucial in order to achieve effective description and classification [18].

Among existing pattern description and classification solutions, the ones based on graphs are particularly successful [9], [12], [14]. In these models, usually patterns are represented as graphs where nodes correspond to local features and edges describe their spatial and geometric

relationships. Then a form of graph matching technique is followed to reach the final goal. This framework has been successfully applied to many computer vision and pattern recognition problems such as feature matching [7], shape analysis [38], object categorization [12], face alignment [48], action recognition [47] etc. This success is mainly due to the ability to encode relationships between different inter/intra class object entities and efficient design of graph-based algorithms.

However, a disadvantage of graphs, when compared to feature vectors, is the significantly increased complexity of many graph-based algorithms. For example, the comparison of two feature vectors for identity can be accomplished in linear time with respect to the length of the two vectors. For the similar operation on generic graphs, *i.e.*, testing two graphs for isomorphism, only exponential algorithms are known as of today. Another serious limitation in the use of graphs for pattern recognition tasks is the incompatibility of most of the mathematical operations in graph domain. For example, computing the (weighted) sum or the product of a pair of entities (which are elementary operations required in many classification and clustering algorithms) is not possible in the domain of graphs, or is atleast not defined in a standardized way. But these operations need to be defined in a particular way for employing machine learning algorithms. A possible way to address this issue is either to define an *explicit embedding* function $\varphi : \mathbb{G} \rightarrow \mathbb{R}^n$ in arbitrary graph domains to a real vector space or to define an *implicit embedding* function $\varphi : \mathbb{G} \rightarrow \mathcal{H}$ in arbitrary graph domains to high dimensional Hilbert space \mathcal{H} where a dot product defines similarity between two graphs $K(i, j) = \langle \varphi(g_i), \varphi(g_j) \rangle$, $g_i, g_j \in \mathbb{G}$. In graph domain, this implicit embedding function is termed as *graph kernel* which basically defines similarity between two graphs. In

- Anjan Dutta is with the Computer Vision Center, Computer Science Department, Universitat Autònoma de Barcelona, Edifici O, Campus UAB, 08193 Bellaterra, Barcelona, Spain. E-mail: adutta@cvc.uab.es.
- Hichem Sahbi is with the LTCI, CNRS, Télécom ParisTech, Université Paris-Saclay, 46 Rue Barrault, 750013 Paris, France. E-mail: hichem.sahbi@telecom-paristech.fr.

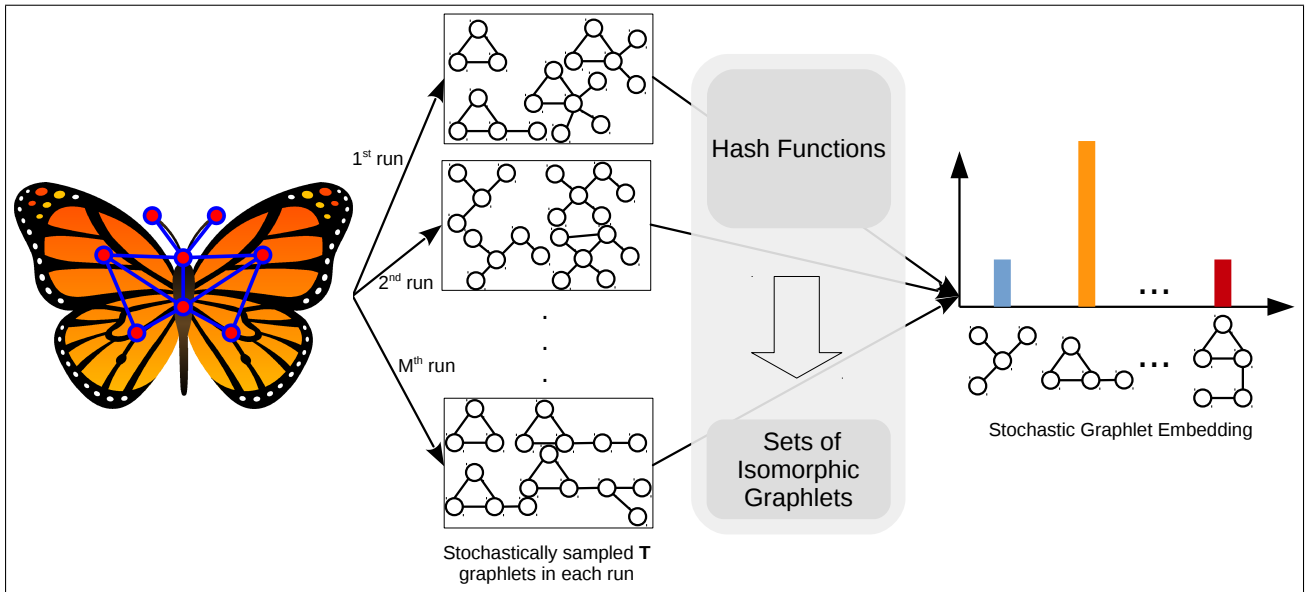


Fig. 1: Overview of our stochastic graphlet embedding (SGE). Given a graph representation (hand crafted for demonstration) G of a pattern (the butterfly in this figure), our stochastic search algorithm is able to sample graphlets of increasing size. Controlled by two parameters M (number of graphlets to be sampled) and T (maximum size of graphlets in terms of number of edges), our method extracts total $M \times T$ graphlets. These graphlets are encoded and partitioned into isomorphic graphlets using our well designed hash functions with low probability of collision. A distribution of different graphlets is obtained by counting the number of graphlets in each of these partitions. This procedure results in a vector space embedding of the graph G , which we call stochastic graphlet embedding.

the next stage, it is usually coupled with machine learning and inference techniques such as support vector machine (SVM) in order to achieve classification. In this context, several authors have already proposed graph matching techniques, to design similarity functions by minimizing pairwise distortions while establishing correspondences between data such as object parts or small image regions [18], [21]. Mainly, graph kernels are designed in two ways. First, approximate graph matching, *i.e.*, by defining similarity between two graphs proportionally to the number of aligned sub-patterns such as nodes, edges, random walks [15], [17], shortest paths [13], cycles [19], subtrees [40] etc. Second, by considering similarity as a decreasing function of the distance between first or high order statistics of their common substructures such as graphlets [35], [39] or graph edit distances of a graph to predefined set of prototype graphs [5]. Thus, the second family of methods first defines an explicit graph embedding and then compute similarities in the embedded vector space. Usually these methods are memory and time demanding as sub-patterns are usually taken from large dictionaries and searched by handling the laborious subgraph isomorphism problem [28] which is known to be NP-complete for general and unconstrained graph structures.

In this paper, we introduce high order *stochastic graphlet embedding* (SGE), which can encode distribution of unlim-

itedly high order¹ connected graphlets or subgraphs in a given graph. The proposed method gathers advantages of two aforementioned families of graph kernels while discarding their limitations. Our technique does not maintain predefined dictionaries of graphlets, and does not perform laborious exact search of these graphlets using subgraph isomorphism. Alternatively, the algorithm generates high order subgraphs or graphlets in a stochastic way, which are much more discriminative than simple walks or tree patterns. The proposed procedure is effective, tractable and is achieved by:

- Significantly restricting graphlets to include only subgraphs belonging to training and test data.
- Parsing a subset of all graphlets, using an efficient stochastic depth first search procedure that extracts statistically meaningful distributions of graphlets.
- Indexing these graphlets with hash functions with low probability of collision, that preserve isomorphic relationships between graphlets quite accurately.

This entire technique randomly samples unlimitedly high order graphlets in a given graph and counts them in an efficient way to obtain their distribution. The procedure is effective and can fetch the distribution of unlimited ordered graphlets with a controlled complexity. Graphlets with relatively high orders have positive and more influencing

1. In general, order of a graph is defined as the total number of vertices present in that graph. In this paper, we alternatively use the term “order” to mean the size of a graph/subgraph/graphlet. This is because the size of our extracted graphlets is not regulated by the number of nodes but the number of edges.

impact on the performance of pattern classification, which has been supported by detailed experimental results.

Fig. 1 illustrates the key idea of our proposed stochastic graphlet embedding algorithm. Here we consider the image of butterfly as a pattern and this pattern has been given a hand crafted graph representation as shown in the figure. Here it is to be noted that without loss of generality our method is applicable to other pattern graph as well. We sample $M \times T$ connected graphlets of increasing size with the proposed stochastic depth first search procedure (to be described later). We also design efficient hash functions with low probability of collision for graphs. After having the graphlets sampled, we partition them into disjoint isomorphic sets with the help of well crafted hash functions. This allows us to count the isomorphic graphlets in a pattern graph, which produces a distribution of graphlets, that we call *stochastic graphlet embedding* (SGE). Thus, our contributions are twofold:

- 1) We have proposed a stochastic depth first search strategy which can parse any given graph and results in desired number of graphlets with specific number of edges to be stated by the user.
- 2) We have found or designed efficient graph hash function with low probability of collision, which can partition sampled graphlets into isomorphic sets quite accurately.

Two most similar works to ours are proposed by Shervashidze *et al.* [39] and Saund [35]. Among them, Shervashidze *et al.* [39] work with a fixed dictionary of subgraphs (upto the graphlets with number of nodes equal to 5). They provide two schemes to enumerate the graphlets, one based on sampling and the second one specifically designed for bounded degree graphs. In both the cases, they restricted to enumerate graphlets with number of nodes only equal to 5. But the enumeration of bigger graphs carries more robust information, which has been revealed in our experiment. On the other hand, Saund [35], given a set of primitives, creates a graph lattice in a bottom-up way. Then this graph lattice is used to enumerate the subgraphs while parsing a given graph. In this way, consideration of limited number of primitives has made the method application specific. In addition, increment of the average degrees of node in a dataset would result in a very big graph lattice, which will increase the time complexity for parsing a graph. In contrast, our method does not maintain a fixed vocabulary of graphlets. The candidate graphlets to be considered for enumeration are entirely determined by the algorithm and have to be present in the training or test set. Our method is not dependent on any specific application and is versatile. This fact has been proven by experiments on different type of datasets, *viz.*, protein structures, chemical compound, form documents, graph representation of digits, shape etc.

The rest of this paper is organized as follows: In Section 2, we review the related works on graph kernel and graph explicit embeddings. Section 3 introduces our efficient stochastic graphlet parsing algorithm, while Section 4 describes hashing techniques to build our stochastic graphlet

embedding. Section 5 presents a detailed experimental validation of the proposed method showing the positive impact of high order graphlets on the performance of graph classification. Finally, Section 6 concludes the paper while briefly providing possible extensions for a future work.

2 RELATED WORKS

One efficient and systematic way to deal the lack of algorithmic tools in graph domain is *graph embedding*. There are two different ways available to embed a graph to high dimensional space. First, mapping a graph to a point in high dimensional Hilbert space where the similarity between two graphs can be computed as a dot product of two points, which is called as *graph implicit embedding* or *graph kernel*. Second, mapping a graph to a point in real vector space and execute all relevant algorithms there, which is termed as *graph explicit embedding*. In following sections, we review relevant works on these two different graph embeddings respectively.

2.1 Graph Kernel

Kernel methods have received enormous popularity during the last few decades. On one hand, this is because of its extendibility of basic linear algorithms to complex non linear ones in a unified and attractive way. Hence non-linear regularities in data are elegantly coped with kernel methods. It is theoretically evident that under some specific conditions, kernel methods are more powerful for difficult pattern recognition tasks than conventional methods [42]. On the other hand, kernel methods make common algorithms available for complex data such as graphs, strings or trees. Therefore, much more efforts have been put to propagate kernel methods to structural domains.

According to the definition, any graph kernel function κ can be regarded as a dot product $\langle \cdot, \cdot \rangle$ of two elements in some high order feature space \mathcal{H} , which is also termed as *kernel machines*. Since in a kernel, dot product is used in the Hilbert space to define the similarity between any two graphs, one can simply compute the kernel function in \mathbb{G} . This procedure is commonly called as *kernel trick* [36] which has a huge impact in practice and allows one to perform mathematical operations on graphs, that can not be computed directly in graph domain. From a higher level perspective, the kernel trick allows one to run algorithms (kernel machines) in implicitly existing feature spaces \mathcal{H} without computing the mapping $\varphi : \mathbb{G} \rightarrow \mathcal{H}$ and even without knowing \mathcal{H} . Below we present three different classes of graph kernels.

2.1.1 Diffusion Kernel

Diffusion kernels are defined with respect to a base similarity function which is used to construct a kernel matrix $\mathbf{K} = (\kappa_{ij})_{N \times N}$ [22], [41]. Here the base similarity function needs to be symmetrical in order to guarantee the kernel matrix to be positive definite. Given a set of graphs $\mathbb{G} = \{g_1, g_2, \dots, g_n\}$, a decay factor $0 < \lambda < 1$, and a similarity measure $s : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{R}$, the $n \times n$ matrix

$\mathbf{S} = (s_{ij})_{N \times N}$ containing the pairwise similarities s_{ij} can be turned into a positive definite kernel [22] by

$$\mathbf{K} = \sum_{k=0}^{\infty} \frac{1}{k!} \lambda^k \mathbf{S}^k = \exp(\lambda \mathbf{S})$$

or the *von Neumann diffusion kernel* [20] defined by

$$\mathbf{K} = \sum_{k=0}^{\infty} \lambda^k \mathbf{S}^k$$

The decay factor λ should be sufficiently small so that it assures the weighting factor λ^k will be negligibly small for sufficiently large k . Therefore, only a finite number of addends in the diffusion kernel sum have to be evaluated in practice. Any graph (dis) similarity measure can be used to build a diffusion kernel for graphs.

2.1.2 Convolution Kernel

A pioneering contribution to the field of graph kernels is the work on *convolution kernels*, which provides a general framework for dealing with complex objects [44]. Convolution kernels deduce the similarity of composite objects from the similarity of their parts. The rationale behind this approach is that a similarity function might be more easily defined or more efficiently be computed for smaller parts rather than for a whole composite object. Given the similarities between the simpler parts of the underlying objects, a convolution operation is applied in order to turn them into a kernel function.

It is evident that graphs are composite objects, as they consist of nodes and edges. The concept of decomposing a graph into smaller parts can be mathematically denoted by a relation R , where $R(g_1, \dots, g_d, g)$ represents the decomposition of g into parts (g_1, \dots, g_d) . By $R^{-1}(g) = \{(g_1, \dots, g_d) : R(g_1, \dots, g_d, g)\}$, we denote the set of decompositions of the graph $g \in \mathbb{G}$. For the definition of convolution kernel, a kernel function κ_i is required for each pair of parts of decomposition $\{g_i\}_{1 \leq i \leq d}$. The convolution kernel function for graphs $g, g' \in \mathbb{G}$ can then be written as follows:

$$\kappa(g, g') = \sum_{(g'_1, \dots, g'_d) \in R^{-1}(g')} \sum_{(g_1, \dots, g_d) \in R^{-1}(g)} \prod_{i=1}^d \kappa_i(g_i, g'_i)$$

Hence, this graph kernel derives the similarity between two graphs g and g' from the sum, over all decompositions, of the similarity products of the parts of g and g' [30].

2.1.3 Substructure Kernel

A third class of graph kernels is based on the analysis of substructures, such as random walks [43], backtrackless walks [1], shortest paths [4], subtrees [40], graphlets [39] in graphs. These kernels measure the similarity of two graphs by counting the number of underlying substructures in both graphs that have all or some labels in common [4]. Among the above mentioned graph kernels, random walk graph kernel has received a lot of attention [15], [43]. In [15], Gärtner *et al.* showed that the number of matching walks

in two graphs G and G' can be computed by means of the direct product graph $G \times G'$, without explicitly enumerating the walks and matching them. This allows one to consider random walks of unlimited length.

2.2 Graph Explicit Embedding

Another approach for surpassing the lack of algorithmic tools for graphs is graph explicit embedding into vector spaces. The main aim of graph embedding is similar to that of graph kernels, *i.e.*, one wants to benefit from both the universality of graphs for pattern representation and the computational convenience of vector spaces for pattern recognition. In contrast to graph kernels, however, graph explicit embedding procedures result in an exclusive embedding of graphs from some arbitrary graph domain \mathbb{G} into a real vector space \mathbb{R}^n . Formally, a graph embedding is a function $\varphi : \mathbb{G} \rightarrow \mathbb{R}^n$ that maps graphs from an arbitrary graph domain \mathbb{G} to a real vector space. Based on the resulting graph maps, the considered pattern recognition task is carried out. Hence the whole weaponry of algorithmic tools originally developed for vectorial data becomes available to graphs.

The graph embedding algorithms can be categorized into three main groups. The first group is based on the frequencies of appearance of some specific substructures, capturing topological information and content of graph through its node and edge labels. This group is known as *graph probing* based methods [26]. For example, in [40] the vector is built by counting the number of non-isomorphic graphlets included in the target graph. The approach proposed by Gibert *et al.* [16] is based on different statistics on the node labels and the edge relations among them. Luqman *et al.* [26] consider the graph information in several levels of topology, structure and attributes. In [35], given a set of primitive nodes, Saund proposed a way to create a graph lattice in a bottom-up way. Later this lattice is used to count number of graphlets to get their distribution in a particular document represented as graph, which is used for similar document retrieval problem. The second group of graph embedding methods is based on *spectral graph theory* [6], [34], [46]. Spectral graph theory is concerned with understanding how the structural properties of graphs can be characterized using eigenvectors of the adjacency or Laplacian matrix [46]. Although graph spectralization exhibits interesting properties which can be used for vector space embedding of graphs, this approach is limited in the sense that spectral methods are not fully able to cope with noisy graphs. This limitation stems from the fact that the eigen decomposition is sensitive towards structural errors, such as missing or spurious nodes. Moreover, spectral methods are applicable to unlabelled graphs or labelled graphs with constrained label alphabets only, although recent work attempts to overcome this limitation [24]. The third group is inspired by *dissimilarity representations* proposed in [32]. Bunke and Riesen present the vectorial description of a graph by its distances to a number of preselected prototype graphs [5]. Recently, regardless of

these above three categories of graph explicit embedding, Mousavi *et al.* [29] have proposed a generic framework with graph pyramid which can hierarchically embed any given graph to a real vector space, which constructs an informative description containing both local and global features.

2.3 Graph Kernel and Explicit Embedding

Graph explicit embedding is closely related to graph kernel. The graph embedding paradigm established by the mapping $\varphi : \mathbb{G} \rightarrow \mathbb{R}^n$ constitutes the foundation of a new graph kernel. Given a graph embedding $\varphi : \mathbb{G} \rightarrow \mathbb{R}^n$, one can define a valid graph kernel κ by computing the standard dot product of two graph maps in the resulting vector space:

$$\kappa(g, g') = \langle \varphi(g), \varphi(g') \rangle$$

Of course, not only the standard dot product but any valid kernel function defined for vectors can be used for the purpose. For instance, an RBF kernel function:

$$\kappa_{RBF}(g, g') = \exp(-\gamma \|\varphi(g) - \varphi(g')\|^2)$$

where $\gamma > 0$ can thus be applied to graph maps or histogram intersection kernel function as follows:

$$\kappa_{HI}(g, g') = \sum \min(\varphi(g), \varphi(g'))$$

We refer to this procedure as *graph embedding kernel* using some specific vector kernel function.

3 HIGH ORDER STOCHASTIC GRAPHLETS

Our goal is to introduce a novel explicit graph embedding technique that combines the representational power of graphs with the robustness of high order graphlets and efficiency of graph hashing. As shown subsequently, patterns represented with graphs are described with distribution of high order graphlets, where these graphlets are extracted using an efficient *stochastic depth first search strategy* and partitioned into isomorphic sets of graphlets using well defined graph hash function.

3.1 Graphs and Graphlets

Let us consider a finite collection of m patterns $\mathcal{S} = \{\mathcal{P}_1, \dots, \mathcal{P}_m\}$. A given pattern $\mathcal{P} \in \mathcal{S}$ can be described with an *attributed graph* which is basically a 4-tuple $G = (V, E, \phi, \psi)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. The two mappings $\phi : V \rightarrow \mathbb{R}^m$ and $\psi : E \rightarrow \mathbb{R}^n$ respectively assign attributes to the nodes and edges of G . An attributed graph $G' = (V', E', \phi', \psi')$ is said to be a *subgraph* of the graph G and is denoted by $G' \subseteq G$ if the following conditions are satisfied:

- $V' \subseteq V$
- $E' = E \cap V' \times V'$
- $\phi'(u) = \phi(u), \forall u \in V'$
- $\psi'(e) = \psi(e), \forall e \in E'$

A graph is said to be *connected* when there is a path between every pair of vertices of it.

A *graphlet* refers to any subgraph g of G that may also inherit the topology and attributes of G . In this work, we only consider *connected graphlets* and in this paper, unless otherwise mention, by graphlets we refer only to the connected ones. We use these graphlets to extract robust structural information in a graph representing a pattern. These graphlets may correspond to actual pattern, their repeated parts, as well as their spatial relationships. Our method neither requires laborious enumeration of graphlet dictionaries nor any execution of subgraph isomorphism for checking isomorphic and non-isomorphic subgraphs.

Algorithm 1 STOCHASTIC-GRAPHLET-PARSING(G): Create a set of graphlets \mathbf{S} by traversing G .

Require: $G = (V, E), M, T$

Ensure: \mathbf{S}

```

1:  $\mathbf{S} \leftarrow \emptyset$ 
2: for  $i = 1$  to  $M$  do
3:    $u \leftarrow \text{SELECTRANDOMNODE}(V)$ 
4:    $U_0 \leftarrow u, A_0 \leftarrow \emptyset$ 
5:   for  $t = 1$  to  $T$  do
6:      $u \leftarrow \text{SELECTRANDOMNODE}(U_{t-1})$ 
7:      $v \leftarrow \text{SELECTRANDOMNODE}(V) : (u, v) \in E \setminus A_{t-1}$ 
8:      $U_t \leftarrow U_{t-1} \cup \{v\}, A_t \leftarrow A_{t-1} \cup \{(u, v)\}$ 
9:      $\mathbf{S} \leftarrow \mathbf{S} \cup \{(U_t, A_t)\}$ 
10:  end for
11: end for

```

3.2 Stochastic Graphlet Parsing

Considering a large graph $G = (V, E, \phi, \psi)$ correspond to a given pattern $\mathcal{P} \in \mathcal{S}$, our goal is to obtain the distribution of high order graphlets in G , without considering a predefined dictionary of them and without explicitly tackling the sub-graph isomorphism problem. The way we acquire graphlets is stochastic and here we not only consider small graphlets but also the higher order ones without constraining their topology or structural properties such as max degree, max number of nodes, etc.

Our graphlet extraction procedure is based on a random walk process that efficiently parses and extracts desired amount of subgraphs from G with increasing complexities in terms of number of edges. The graphlet extraction process outlined in Algorithm 1 is iterative and regulated by two parameters M and T , where M denotes the number of runs or the number of distinct connected graphlets to be extracted and T denotes the maximum number of edges each graphlet should possess. M is set to relatively large values in order to make graphlet generation statistically meaningful (see Line 2). Our stochastic graphlet parsing algorithm iteratively visits the connected nodes and edges in G and extracts or samples different graphlets with an increasing number of edges denoted as $t \leq T$ (see Line 5), following a T -step random walk process with restart. Considering U_t, A_t respectively as the aggregated sets of visited nodes and edges till step t , we initialize, $A_0 = \emptyset$ and

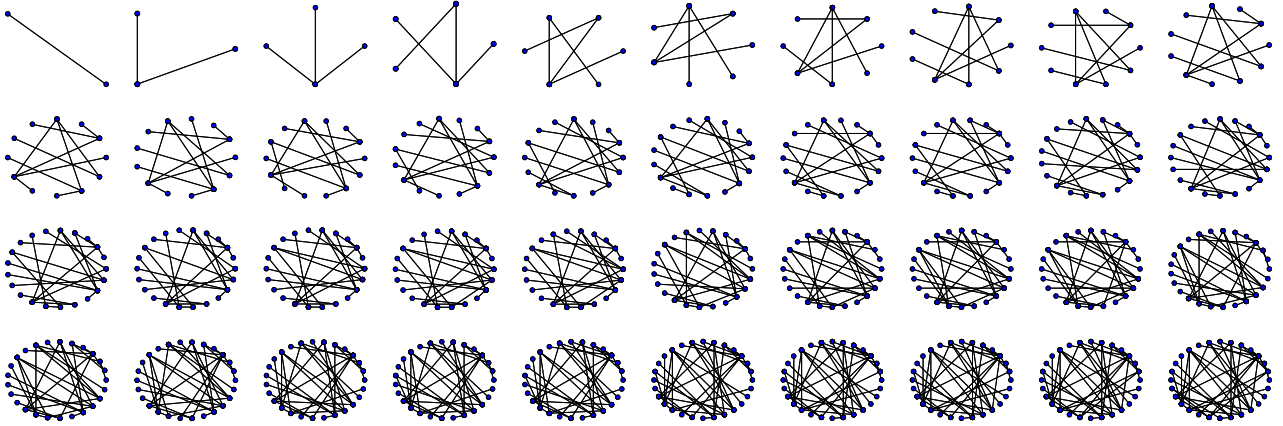


Fig. 2: Example of graphlets with an increasing number of edges, for generating these particular examples we have used $T = 40$. This is to show that our stochastic search algorithm is not restricted to any size bound.

U_0 with a randomly selected node u which is uniformly sampled from V (see Line 3 and Line 4). For $t \geq 1$, the process continues by sampling a subsequent node $v \in V$, according to the following distribution

$$P_t(v|u) = \alpha P_{t,w}(v|u) + (1 - \alpha) P_{t,r}(v)$$

where α is the probability of restarts, which is set to $\frac{1}{2}$ in practice, $P_{t,w}(v|u)$ corresponds to the conditional probability of a random walk from node u to its neighbour v , and $P_{t,r}(v)$ is the probability to restart the random walk, from an already visited node $v \in U_{t-1}$, defined as

$$P_{t,r}(v) = 1_{\{v \in U_{t-1}\}} \cdot \frac{1}{|U_{t-1}|}$$

with $1_{\{ \}}$ being the indicator function. Hence, node sampling is achieved in one of the following ways:

- $P_{t,w}$ (random walks): in order to expand a currently generated graphlet with neighbours of u that possibly have similar visual features/attributes.
- $P_{t,r}$ (restarts): in order to continue the expansion of the currently generated graphlet using other nodes if the set of edges connected to u is fully exhausted.

Finally, if $(u, v) \in E$ and $(u, v) \notin A_{t-1}$, then the aggregated sets of nodes and edges at step t are updated as:

$$U_t \leftarrow U_{t-1} \cup \{v\}$$

$$A_t \leftarrow A_{t-1} \cup \{(u, v)\}$$

which is shown in Line 8. It is to be noted that the Algorithm 1 iterates M times and in each iteration it generates a graphlet with number of edges equal to T . From that, one can evidently extract T graphlets respectively with edges $1, \dots, T$. Hence, with two parameters M and T , our stochastic graphlet parsing algorithm results in total $M \times T$ graphlets. As it is obvious, Algorithm 1 is highly parallelizable and hence efficient enough to extract huge number of graphlets from a graph G .

The actual number of samples needed to achieve a given confidence with a small probability of error is called the

sample complexity. The way of determining M is done according to the Theorem 1.

Theorem 1. Let D be a probability distribution on a finite set of cardinality a . Let $X := \{X_j\}_{j=1}^M$, with $X_j \sim D$. For a given error $\epsilon > 0$ and confidence $\delta > 0$,

$$M = \left\lceil \frac{2 \left(a \ln 2 + \ln\left(\frac{1}{\delta}\right) \right)}{\epsilon^2} \right\rceil$$

samples suffice to ensure that $P\left\{ \|D - \hat{D}^M\| \geq \epsilon \right\} \leq \delta$

TABLE 1: Sample complexity for graphlets having number of edges ranging from 1 to 10 for a given pair of ϵ and δ .

Size of graphs	Number of possible graphs (a)	M ($\epsilon = 0.1, \delta = 0.1$)	M ($\epsilon = 0.1, \delta = 0.05$)	M ($\epsilon = 0.05, \delta = 0.1$)	M ($\epsilon = 0.05, \delta = 0.05$)
1	1	600	738	2397	2952
2	1	600	738	2397	2952
3	3	877	1016	3506	4061
4	5	1154	1293	4615	5170
5	12	2125	2263	8497	9051
6	30	4620	4759	18478	19033
7	79	11413	11551	45649	46204
8	227	31930	32069	127718	128273
9	710	98888	99027	395550	396105
10	2322	322359	322497	1289433	1289987

The proof of the above theorem is out of the main scope of this paper and further details can be found in [39], [45]. However, for having a better understanding on the Theorem 1, we have constructed the Table 1 where we have enumerated the values of M for different sets of graphlets in terms of number edges, pairs of confidence and error values. For example, there are only 5 graphlets² with number of edges equal to 4. Hence for a given $\epsilon = 0.1$ and $\delta = 0.1$, the values of M will be equal to 1154. And the values of M will respectively be 1293, 4615 and 5170 for $(\epsilon = 0.1, \delta = 0.05)$, $(\epsilon = 0.05, \delta = 0.1)$ and $(\epsilon = 0.05, \delta = 0.05)$.

2. Refer to the article A002905 (<http://oeis.org/A002905>) of OEIS (Online Encyclopedia of Integer Sequence) to know more about the number of graphs with a specific number of edges.

4 HASHED GRAPHLETS STATISTICS

In order to obtain the distribution of parsed graphlets in a given graph G , it is needed to identify different sets of isomorphic graphlets from the sampled ones. To do that it becomes useless to tackle subgraph isomorphism, which is again an NP-complete problem for general graphs [28]. Instead one may partition the extracted graphlets into subsets, each of which includes only isomorphic graphlets. Yet this process is also computationally intractable and known to be GI-complete, and no polynomial solution is known for general graphs. In what follows, we approach the problem differently using graph hashing. The latter generates compact and also effective hash codes for graphlets based on their local as well as holistic topological characteristics and allows one to group generated isomorphic graphlets while colliding non-isomorphic ones with a very low probability.

The goal of our graphlet hashing is to assign extracted graphlets of a given graph G to corresponding subsets of isomorphic graphlets (a.k.a. histogram bins) in order to count and quantify their distributions (see Algorithm 2 and Line 9). For that purpose, we maintain a global hash table **HashTable** whose entry corresponds to a compact vector representation (*i.e.* hash code) resulting from the output of a hash function which measures the topological characteristics of graphlets (Line 4) (see [11] for a detailed discussion of these topological characteristics). Now to allocate a given graphlet to its corresponding histogram bin, its hash code is mapped to the index of the global hash table **HashTable** (see Line 8), whose corresponding hash code gives a hit with the hash code in question. If a given hash code does not exist in the global hash table at some point, we consider it as a new hash code (and hence the corresponding graphlet as a new graphlet) encountered by the system and append it at the end of the hash table **HashTable**. When using appropriate hash functions (see Section 4.1), this algorithm, even though not tackling the subgraph isomorphism, is able to count the number of isomorphic subgraphs in a given graph with a controlled (polynomial) complexity.

Algorithm 2 HASHED-GRAPHLETS-STATISTICS(G): Create a histogram **H** of graphlet distribution for a graph G .

Require: G

Ensure: **H**

```

1: S  $\leftarrow$  STOCHASTIC-GRAPHLET-PARSING( $G$ )
2: HashTable  $\leftarrow$   $\emptyset$ , H $i$   $\leftarrow$  0,  $i = 1, \dots, |\mathbf{S}|$ 
3: for all  $g \in \mathbf{S}$  do
4:   hashcode  $\leftarrow$  HASHFUNCTION( $g$ )
5:   if hashcode  $\notin$  HashTable then
6:     HashTable  $\leftarrow$  HashTable  $\cup$  {hashcode}
7:   end if
8:    $i \leftarrow$  GETINDEX-IN-HASHTABLE(hashcode)
9:   H $i$   $\leftarrow$  H $i$  + 1
10: end for

```

There are two types of hash functions exist for graphs: *local* and *holistic*. Holistic functions are computed globally on a given graphlet and include number of nodes/edges, sum/product of node labels, and frequency distribution of

node labels, while local functions are computed at the node level. Some local hash functions of graphs can be as follows:

- *Local clustering coefficient* of a node u in a graph is the ratio between the number of triangles connected to u and the number of triples centred around u . The local clustering coefficient of a node in a graph quantifies how close its neighbours are for being a clique.
- *Betweenness centrality* of a node u is the number of shortest paths from all nodes to all others that pass through the node u . In a generic graph, betweenness centrality of node gives a measurement about the centrality of that node with respect to the entire graph.
- *Core number* of a node u is the largest integer c such that the node u has degree greater than zero when all the nodes of degree less than c are removed.
- *Degree* of a node u is the number of edges connected to the node u .

As these local measures are sensitive to the ordering of nodes in a given graphlet, we sort and concatenate them in order to obtain a global permutation invariant hash code.

4.1 Hash Function Selection

An *ideal and reliable* hash function is expected to provide identical hash codes for two isomorphic graphlets and two different hash codes for two non-isomorphic ones. While it is easy to design hash functions that provide identical hash codes for isomorphic graphlets, it is very challenging to guarantee that non-isomorphic graphlets could never be mapped to the same hash code. This is also in accordance with the fact that graph isomorphism detection is GI-complete and no polynomial algorithm is known to solve it. The possibility of mapping two non-isomorphic graphlets to the same hash code is termed as *probability of collision*. Let f be a function that returns a hash code for a given graphlet, then the probability of collision of that function can be defined as

$$E(f) = P((g, g') \in H_0 \mid f(g) = f(g'))$$

here g, g' denote two graphlets, and the probability is with respect to H_0 which stands for pairs of non-isomorphic graphlets. Equivalently, we can define H_1 as the pairs of isomorphic graphlets, and since the cardinality of H_0 is really huge for graphlets with large number of edges, *i.e.*, $|H_1| \ll |H_0|$, one may instead consider

$$E(f) = 1 - P((g, g') \in H_1 \mid f(g) = f(g'))$$

which also results from the fact that our hash functions produce same codes for isomorphic graphlets. For bounded t ($t \leq T$), the evaluation of $E(f)$ becomes tractable and reduces to

$$E(f) = 1 - \frac{\sum_{g, g'} 1_{\{(g, g') \in H_1\}}}{\sum_{g, g'} 1_{\{f(g) = f(g')\}}}$$

Considering a collection of hash functions $\{f_c\}_c$, the best one is chosen as

$$f^* = \arg \min_{f_c} E(f_c)$$

TABLE 2: Probability of collision $E(f)$ of different hash functions viz. *betweenness centrality*, *core numbers*, *degree* and *clustering coefficients*. These values are enumerated on graphlets with number of edges $t = 1, \dots, 10$; some examples of these graphlets are shown in Fig 2.

Size of graphs (t)	Number of possible graphs (n)	Number of comparisons for checking collisions (${}^n C_2$)	betweenness centrality		core numbers		degree		clustering coefficients	
			Number of collision occurs	Probability of collision	Number of collision occurs	Probability of collision	Number of collision occurs	Probability of collision	Number of collision occurs	Probability of collision
1	1	—	0	0.00000	0	0.0000	0	0.0000	0	0.0000
2	1	—	0	0.00000	0	0.0000	0	0.0000	0	0.0000
3	3	3	0	0.00000	1	0.3333	0	0.0000	1	0.3333
4	5	10	0	0.00000	2	0.2000	0	0.0000	3	0.3000
5	12	66	0	0.00000	7	0.1061	2	0.0303	7	0.1061
6	30	435	0	0.00000	22	0.0506	11	0.0253	18	0.0414
7	79	3081	1	0.00032	68	0.0221	44	0.0143	50	0.0162
8	227	25651	5	0.00019	211	0.0082	167	0.0065	157	0.0061
9	710	251695	27	0.00011	687	0.0027	604	0.0024	537	0.0021
10	2322	2694681	108	0.00004	2290	0.0008	2145	0.0008	1907	0.0007

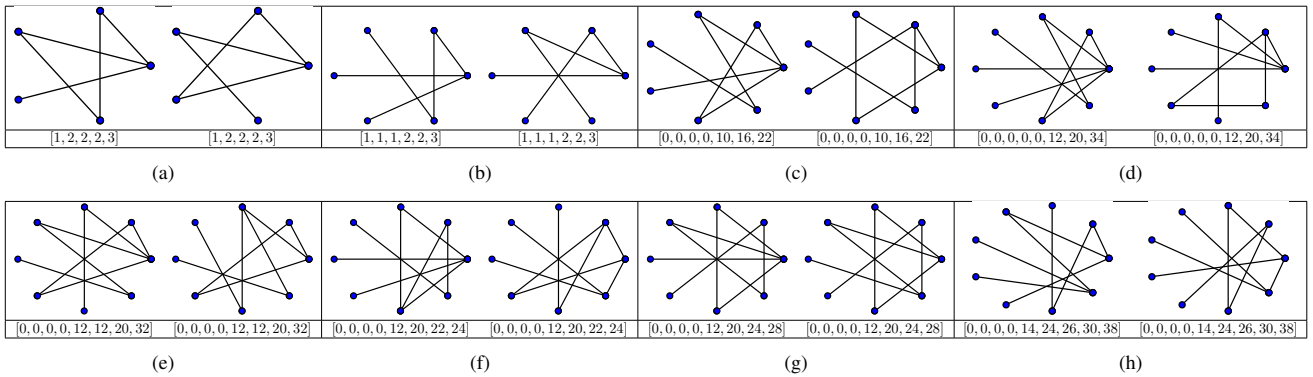


Fig. 3: Example of non-isomorphic graphlets having same hash code (shown just below the respective graphlets) with different hash functions: (a)-(b) Two pairs of non-isomorphic graphlets with $t = 5$, that have same degree values, (c) A pair of non-isomorphic graphlets with $t = 7$, that have same betweenness centrality values, (d)-(h) Five pairs of non-isomorphic graphlets with $t = 8$, that have same betweenness centrality values.

For getting an idea on different hash functions based on the probability of collision, we have created the Table 2 which shows the values of $E(f)$ for different hash functions such as *betweenness centrality*, *core numbers*, *degree* and *clustering coefficients*, and for different graphlet sizes in terms of number of edges ranging from 1 to 10. To construct the table, we enumerate all the non-isomorphic graphs having number of edges less or equal to 10 and compute the hash codes with the above mentioned hash functions to quantify the number and probability of collisions. From what we observe that $E(f) \rightarrow 0$ as $t \rightarrow \infty$, for all the hash functions f . Moreover, the hash function *degree* of nodes has probability of collision equal to 0 for graphlets with $t \leq 4$ but has increased for $t \geq 5$, while *betweenness centrality* has lowest probability of collision for all t . The number of non-isomorphic graphs with the same *betweenness centrality* is very less for smaller sized graphs and increases in a very low rate with the increment of size (see Fig. 3 for examples). This is also a well known fact within the network analysis community that two graphs with the same *betweenness centrality* would indeed be isomorphic with high probability [8], [31]. In pattern recognition domain, to the best of our knowledge, we are the first one to highlight this fact for graph isomorphism detection. Since this fact is quite unknown to the community, we

have provided a MATLAB tool³ to generate the particulars shown in Table 2. In Table 2, we have only provided the details for graphs with edges less than or equal to 10, as the number of graphs beyond that is very high and result enumerating data with them is time and space consuming. However, it is to be noted that an advantage of *degree* over *betweenness centrality* is its computational complexity. In order to trade off probability of collision and computation time, in practice we consider *degree* for graphlets with $t \leq 4$ and the *betweenness centrality* for graphlets with larger t .

The hash functions mentioned above basically give a signature about the structure or topology of a graphlet and do not consider the node/edge attributes or labels of the graphlets, which is absolutely fine as long as we consider classifying the unlabelled graphs. Here it is to be mentioned that our method is also capable to handle the classification problem of labelled graphs that have a small set of node/edge attributes. For that, we create node/edge signature of the underlying graphlets by arranging the node/edge attributes according to the sorted index obtained while sorting the hash code values mentioned in Section 4. These node/edge signatures are concatenated with the

3. Available at <https://github.com/AnjanDutta/StochasticGraphletEmbedding/tree/master/HashFunctionGraphlets>

sorted hash values to obtain a holistic hash code for a labelled graphlet.

COMPUTATIONAL ANALYSIS

The average time complexity of Algorithm 1 is $O(MT)$ and that for computing the hash code for all the extracted graphlets in Algorithm 2 is $O(MTC)$, where M is the number of runs or number of distinct graphlets we want to obtain, T is the maximum number of edges a graphlet should possess and C is the average time complexity of the hash function used. C for hash function degree is $O(|V|)$ and that for betweenness centrality is $O(|V||E|)$. Since the graphlets are sampled independently and after sampling they are independent, both the two processes mentioned above are trivially parallelizable. The checking for the existence of the computed hash codes in the hash table and their appendage can be done with a worst case time complexity of $O(MTN)$, where N is the number of unique hash codes generated by the hash function. Since our algorithm checks the hash table for the already existing hash codes, the last process is sequential. Here it is to be clearly noted that the time taken by the method is not at all dependent on the size of the input graph G , but on the parameters M , T and the hash function we use. For getting an idea about the time required in reality by the proposed method, we mention some examples as follows. With $M = 11413$, $T = 7$, our entire method in a fully serialized framework takes 6.13 seconds on an average for parsing a graph to generate the stochastic graphlets, compute their hash codes and finding their respective histogram bins. While in a parallelized framework with 4 workers, it takes only 1.74 seconds. Furthermore, with $M = 46204$, $T = 7$, in a serialized framework, our method takes 22.57 seconds to execute the entire process, while in a parallelized configuration it takes only 5.62 seconds. Additionally, with $M = 4061$, $T = 3$, in a serialized architecture, our method takes 1.13 seconds to complete the entire procedure, while in a parallelized organization, it takes 1.01 seconds. So from that we can conclude that for larger M and T , the parallelization is effective otherwise it is not, which is mainly because of the overhead caused distributing subtasks to underlying workers.

5 EXPERIMENTAL VALIDATION

In order to evaluate the impact of our proposed stochastic graphlet embedding, we consider five different experiments described below. Here it is to be mentioned that we have evaluated our method both with the distribution of graphlets with a fixed number of edges and also their combination. It has been revealed through our experiments that combination of graphlets having increasing number of edges often improves the performance. However, to save the space in the main manuscript and to make the comparison with different state-of-the-art methods relevant, we only put the related results in this article and the other in the supplementary material provided with the

manuscript. A generic MATLAB source code of our proposed method is available at <https://github.com/AnjanDutta/StochasticGraphletEmbedding>. Moreover, the other experiment specific source codes will be available upon the acceptance of the article.

5.1 MUTAG, PTC, ENZYME and D&D Databases

In this section, we show the impact of our proposed stochastic graphlet embedding on the performance of graph classification using 4 publicly available *unlabelled* graph databases: *MUTAG*, *PTC*, *ENZYME* and *D&D*. The *MUTAG* dataset contains graphs representing 188 chemical compounds which are either mutagenic or not mutagenic. So here the task of the classifier is to predict the mutagenicity of the chemical compounds, which is a two class classification problem. The *PTC* or Predictive Toxicology Challenge dataset consists of graphs of 344 chemical compounds known to cause or not cause cancer in rats and mice. Hence the task of the classifier is to predict the cancerogenicity of the chemical compounds, which is also a two class classification problem. The *ENZYME* dataset contains graphs representing protein tertiary structures consisting of 600 enzymes from the *BRENDA* enzyme. Here the task is to correctly assign each enzyme to one of the 6 EC top level. The *D&D* dataset consists of 1178 graphs of protein structures which are either enzyme or non-enzyme. Therefore, the task of the classifier is to predict if a protein is enzyme or not, which is essentially a two class classification problem. Some details on the above four datasets are shown in Table 3.

TABLE 3: Some details on *MUTAG*, *PTC*, *ENZYME* and *D&D* graph datasets.

Datasets	#Graphs	Classes	Avg. #nodes	Avg. #edges
<i>MUTAG</i>	188	2 (125 vs. 63)	17.7	38.9
<i>PTC</i>	344	2 (192 vs. 152)	26.7	50.7
<i>ENZYME</i>	600	6 (100 each)	32.6	124.3
<i>D&D</i>	1178	2 (691 vs. 487)	284.4	1921.6

Since these datasets are unlabelled, there is only one graphlet both for $t \in \{1, 2\}$. Hence sampling graphlets having number of edges equal to 1 and 2 does not make any discrimination among patterns here. Therefore, we sample graphlets with $t \geq 3$. For obtaining the performance measures, we have executed a 10-fold cross validation scheme and reported the average classification accuracies and respective standard deviations in Table 4. We have also shown comparison against state-of-the-art graph kernels including (i) the standard random walk kernel (RW) [43], that counts common random walks in two graphs, (ii) the shortest path kernel (SP) [4], that compares shortest path lengths in two graphs and (iii) also the graphlet kernel (GK) [39], that compares graphlets with k nodes ($k \in \{3, 4, 5\}$). The results of these state-of-the-art methods are directly taken from [39] that also followed the same experimental protocol as ours. Table 4 shows the impact of our proposed stochastic graphlet embedding for different pairs of ϵ and δ with graphlets having increasing number

TABLE 4: Classification accuracies (in %) on MUTAG, PTC, ENZYME and D&D datasets. RW corresponds to the random walk kernel [43], SP stands for shortest path kernel [4], while “GK Ak m” stands for the classical graphlet kernel [39] computed using all graphlet (not only connected ones) with m samples each with size k . A variant of GK, referred as “GK Cn”, denotes the graphlet kernel computed using all connected graphlets of size k . In these results, “> 1 day” means that results are not available for the state-of-the-art method *i.e.* computation did not finish within 24 hours. SGE refers to our proposed stochastic graphlet embedding.

Kernel	MUTAG	PTC	ENZYME	D & D
RW [43]	71.89 ± 0.66	55.44 ± 0.15	14.97 ± 0.28	> 1 day
SP [4]	81.28 ± 0.45	55.44 ± 0.61	27.53 ± 0.29	> 1 day
GK A3 1016	79.70 ± 0.43	55.34 ± 0.33	23.07 ± 0.38	74.92 ± 0.12
GK A3 1154	79.54 ± 0.41	54.90 ± 0.42	24.22 ± 0.37	75.05 ± 0.10
GK A3 4061	80.91 ± 0.40	55.21 ± 0.34	25.45 ± 0.70	75.23 ± 0.13
GK A3 4615	80.21 ± 0.37	55.13 ± 0.37	24.85 ± 0.79	75.44 ± 0.13
GK A3 all	82.11 ± 0.62	55.26 ± 0.39	25.80 ± 0.23	75.41 ± 0.13
GK C3	66.55 ± 0.83	57.68 ± 0.43	19.80 ± 0.21	74.14 ± 0.12
GK A4 1986	79.80 ± 0.38	58.88 ± 0.50	26.93 ± 0.57	74.47 ± 0.17
GK A4 2125	80.69 ± 0.31	58.86 ± 0.53	27.25 ± 0.44	74.50 ± 0.14
GK A4 7942	81.74 ± 0.44	59.25 ± 0.50	27.96 ± 0.40	74.53 ± 0.16
GK A4 8497	81.48 ± 0.38	59.39 ± 0.57	28.06 ± 0.45	74.59 ± 0.16
GK A4 8497	82.17 ± 0.58	59.65 ± 0.31	28.95 ± 0.50	74.62 ± 0.12
GK C4	69.00 ± 0.74	58.62 ± 0.41	23.61 ± 0.22	75.90 ± 0.10
GK A5 5174	81.62 ± 0.69	58.26 ± 0.47	29.54 ± 0.42	75.11 ± 0.14
GK A5 5313	81.93 ± 0.71	58.29 ± 0.42	29.52 ± 0.25	75.14 ± 0.14
GK A5 20696	82.64 ± 0.66	57.88 ± 0.54	29.96 ± 0.52	75.52 ± 0.12
GK A5 21251	83.27 ± 0.79	58.03 ± 0.42	30.48 ± 0.51	75.35 ± 0.10
GK A5 all	83.50 ± 0.60	58.65 ± 0.40	30.64 ± 0.26	> 1 day
GK C5	70.94 ± 0.76	56.06 ± 0.46	26.66 ± 0.23	> 1 day
SGE ($t = 3, \epsilon = 0.1, \delta = 0.1$)	71.67 ± 0.86	53.53 ± 0.04	24.17 ± 0.54	60.00 ± 0.01
SGE ($t = 3, \epsilon = 0.1, \delta = 0.05$)	75.56 ± 0.52	53.53 ± 0.76	25.33 ± 0.75	60.42 ± 0.23
SGE ($t = 3, \epsilon = 0.05, \delta = 0.1$)	86.11 ± 0.00	54.12 ± 0.48	29.17 ± 0.03	63.39 ± 0.58
SGE ($t = 3, \epsilon = 0.05, \delta = 0.05$)	84.44 ± 0.74	55.88 ± 0.67	29.17 ± 0.10	64.07 ± 0.99
SGE ($t = 4, \epsilon = 0.1, \delta = 0.1$)	77.78 ± 0.41	55.59 ± 0.27	24.00 ± 0.92	59.83 ± 0.23
SGE ($t = 4, \epsilon = 0.1, \delta = 0.05$)	78.89 ± 0.41	60.29 ± 0.39	26.00 ± 0.26	59.92 ± 0.88
SGE ($t = 4, \epsilon = 0.05, \delta = 0.1$)	82.22 ± 0.31	61.18 ± 0.17	30.67 ± 0.85	64.41 ± 0.59
SGE ($t = 4, \epsilon = 0.05, \delta = 0.05$)	81.67 ± 0.89	63.53 ± 0.23	30.17 ± 0.72	64.32 ± 0.24
SGE ($t = 5, \epsilon = 0.1, \delta = 0.1$)	86.11 ± 0.05	56.18 ± 0.26	30.50 ± 0.43	65.76 ± 0.60
SGE ($t = 5, \epsilon = 0.1, \delta = 0.05$)	86.11 ± 0.05	54.71 ± 0.23	30.17 ± 0.46	65.68 ± 0.84
SGE ($t = 5, \epsilon = 0.05, \delta = 0.1$)	85.56 ± 0.52	62.06 ± 0.90	32.17 ± 0.27	68.90 ± 0.22
SGE ($t = 5, \epsilon = 0.05, \delta = 0.05$)	85.00 ± 0.89	62.06 ± 0.79	31.17 ± 0.85	68.64 ± 0.81
SGE ($t = 6, \epsilon = 0.1, \delta = 0.1$)	87.78 ± 0.31	59.41 ± 0.06	28.67 ± 0.22	68.98 ± 0.90
SGE ($t = 6, \epsilon = 0.1, \delta = 0.05$)	88.33 ± 0.15	61.47 ± 0.52	28.50 ± 0.66	70.08 ± 0.48
SGE ($t = 6, \epsilon = 0.05, \delta = 0.1$)	88.89 ± 0.70	57.65 ± 0.58	36.33 ± 0.28	72.63 ± 0.37
SGE ($t = 6, \epsilon = 0.05, \delta = 0.05$)	89.75 ± 0.24	55.59 ± 0.96	35.17 ± 0.26	73.05 ± 0.64
SGE ($t = 7, \epsilon = 0.1, \delta = 0.1$)	85.56 ± 0.68	58.53 ± 0.99	37.33 ± 0.46	72.54 ± 0.66
SGE ($t = 7, \epsilon = 0.1, \delta = 0.05$)	86.11 ± 0.93	57.06 ± 0.82	36.67 ± 0.85	72.80 ± 0.41
SGE ($t = 7, \epsilon = 0.05, \delta = 0.1$)	86.67 ± 0.37	59.12 ± 0.26	40.00 ± 0.50	76.08 ± 0.33
SGE ($t = 7, \epsilon = 0.05, \delta = 0.05$)	87.22 ± 0.27	60.00 ± 0.99	40.67 ± 0.40	76.58 ± 0.27

of edges (see Table 1 for sample size M for different graphlet size and pairs of ϵ and δ). The results show that the increasing size of graphlets generally have a positive impact on the classification performance as the accuracies obtained with bigger graphlets is, in general, better than those obtained with smaller graphlets. This fact is also supported by the results obtained with the classical graphlet kernel [39]. However, our stochastic graphlet embedding has achieved the best results on all the four datasets, which show the effectiveness of considering the distribution of high order graphlets. Moreover, it is to be observed that on D&D dataset, the random walk kernel, the shortest path kernel and the classical graphlet kernel with exact graphlet enumeration scheme take a long runtime (> 1 day), because of the very big size of graphs. Here our proposed method turns out to be very effective as the entire technique is based on random sampling of high order graphs.

5.2 COIL, GREC, AIDS and MAO databases

Another set of experiments are performed on a separate collection of *labelled* graphs. For that, we have considered four different datasets, three of them *viz.* COIL, GREC and AIDS are taken from the IAM graph database repos-

itory⁴ [33] and the other one *i.e.* MAO is taken from the GREYC Chemistry graph dataset collection⁵. The COIL database contains 3900 graphs of 100 different objects each having 39 instances with different angle of rotation. So here the task of the classifier is to correctly predict the object class of the graphs, which will be one of the 100 objects. So this is a 100 class classification problem. The GREC dataset consists of 1100 graphs representing 22 different architectural and electronic symbols each having 50 instances with different levels of noise. Therefore the task of the classifier is to predict the class of the graphs, which will be one of the 22 architectural or electronic symbols. Therefore, it is a 22 class classification task. The AIDS database consists of 2000 graphs representing molecular compounds which are constructed from the AIDS Antiviral Screen Database of Active Compounds⁶. This dataset consists of two classes *viz.* active (400 elements) and inactive (1600 elements), which respectively represent molecules with activity against HIV or not. So the task of the classifier is to correctly predict the activity or inactivity of the graphs against HIV, which is a two class classification problem. The MAO dataset composed of 68 graphs representing molecules that either inhibit the monoamine oxidase which is an antidepressant drugs (38 molecules) or do not inhibit that (30 molecules). Hence the task of the classifier is to predict the inhibiting property of the graphs, which is also a two class classification problem. Some details on these datasets are listed in Table 5.

TABLE 5: Some details on COIL, GREC, AIDS and MAO graph datasets.

Datasets	#Graphs	Classes	Avg. #nodes	Avg. #edges	Node labels	Edge labels
COIL	3900	100 (39 each)	21.5	54.2	NA	Valency of bonds
GREC	1100	22 (50 each)	11.5	11.9	Type of joint: corner, intersection, etc.	Type of edge: line or curve.
AIDS	2000	2 (1600 vs. 400)	15.7	16.2	Label of atoms	Valency of bonds
MAO	68	2 (38 vs. 30)	18.4	19.6	Label of atoms	Valency of bonds

Since these databases are labelled, graphlets with number of edges equal to one would contain distinguishing attributes. Hence we do sample graphlets with $t \geq 1$ for these datasets. For obtaining the performance measures, the train, validation and test sets available with the COIL, GREC and AIDS datasets are considered, and accuracies obtained on the test sets are reported in Table 6. For the MAO database, we have followed the leave one out procedure with a two-class SVM, which is made for each of the 68 graphs of the dataset. Here, it is to be mentioned that all the experimental protocols mentioned above are the standard scheme followed by most of the state-of-the-art methods. Table 6 shows the performance of our proposed stochastic graphlet embedding on these four datasets for different size of graphlets and pairs of ϵ and δ . In this table, we have also shown comparison against the state-of-the-art methods on the above mentioned datasets. Similar to the previous experiment, here also we observe that with

4. Available at <http://www.fki.inf.unibe.ch/databases/i-am-graph-database>

5. Available at <https://brun101.users.greyc.fr/CHEMISTRY/>

6. See at http://dtp.nci.nih.gov/docs/aids/aids_data.html

TABLE 6: Classification accuracies (in %) obtained by our proposed stochastic graphlet embedding (SGE) on COIL, GREC, AIDS and MAO datasets and comparison with state-of-the-art methods *viz.* random walk kernel [43], dissimilarity embedding [5] and node attribute statistics [16].

Method	COIL	GREC	AIDS	MAO
Random Walk Graph Kernel [43]	—	—	98.5	82.4
Dissimilarity Embedding [5]	96.8	95.1	98.1	91.2
Node Attribute Statistics [16]	98.1	99.2	—	—
SGE ($t = 1, \epsilon = 0.1, \delta = 0.1$)	89.60	90.15	94.33	82.35
SGE ($t = 1, \epsilon = 0.1, \delta = 0.05$)	90.60	89.96	93.13	82.35
SGE ($t = 1, \epsilon = 0.05, \delta = 0.1$)	92.40	91.67	93.40	85.29
SGE ($t = 1, \epsilon = 0.05, \delta = 0.05$)	93.90	92.23	94.67	88.24
SGE ($t = 2, \epsilon = 0.1, \delta = 0.1$)	91.50	93.37	94.73	85.29
SGE ($t = 2, \epsilon = 0.1, \delta = 0.05$)	92.40	92.80	95.93	85.29
SGE ($t = 2, \epsilon = 0.05, \delta = 0.1$)	93.90	94.13	96.40	85.29
SGE ($t = 2, \epsilon = 0.05, \delta = 0.05$)	94.40	95.94	96.47	88.24
SGE ($t = 3, \epsilon = 0.1, \delta = 0.1$)	91.80	94.89	96.47	88.24
SGE ($t = 3, \epsilon = 0.1, \delta = 0.05$)	93.70	95.27	97.00	85.29
SGE ($t = 3, \epsilon = 0.05, \delta = 0.1$)	94.70	95.51	96.80	85.29
SGE ($t = 3, \epsilon = 0.05, \delta = 0.05$)	95.90	96.51	96.67	91.18
SGE ($t = 4, \epsilon = 0.1, \delta = 0.1$)	93.50	96.59	96.87	88.24
SGE ($t = 4, \epsilon = 0.1, \delta = 0.05$)	94.70	97.02	97.00	91.18
SGE ($t = 4, \epsilon = 0.05, \delta = 0.1$)	95.80	97.59	97.13	91.18
SGE ($t = 4, \epsilon = 0.05, \delta = 0.05$)	96.50	98.83	97.00	94.12
SGE ($t = 5, \epsilon = 0.1, \delta = 0.1$)	94.90	97.59	97.07	91.18
SGE ($t = 5, \epsilon = 0.1, \delta = 0.05$)	95.50	97.97	97.67	91.18
SGE ($t = 5, \epsilon = 0.05, \delta = 0.1$)	97.90	98.35	98.07	94.12
SGE ($t = 5, \epsilon = 0.05, \delta = 0.05$)	98.40	99.17	97.67	97.06

increment of graphlet size, in general, the performance of the proposed method increases. Moreover in most of the cases, greater number of sampling (lower values of ϵ and δ) of graphlets improves the accuracy of the system, which is justified since with bigger number of samplings, corresponding distribution of graphlets approaches to the actual statistics. Results on COIL and MAO show that our proposed method outperforms the existing state-of-the-art methods. For GREC and AIDS datasets, our method also performed quite close to the existing ones which already performed utterly well on those databases. This experiment shows that similar to the unlabelled databases, our method is also applicable to the labelled datasets with fair amount of attributes.

5.3 AMA Dental Forms database

This section presents the experiments applying our method for the indexing and retrieval of forms documents that contain consistent subgraphical structure. This experiment is inspired by the experiment in the paper [35] and done on the same publicly available benchmark⁷ made available by the aforementioned paper. Since the main aim of both the approaches is to use subgraphs as image features and their distribution as global image description, we believe this experiment is relevant and useful for the comparison of these two methods. The focal problem of this experiment is the indexing and retrieval of forms documents that have sparse and inconsistent textual content due to diverse filling of the data fields. This type of forms usually contain a network of rectilinear rule lines serving as region separators,

data field locators, and field group indicators (see Fig. 4). Document retrieval is a principal step in many document image processing systems and from that perspective our method would find an important real world application.

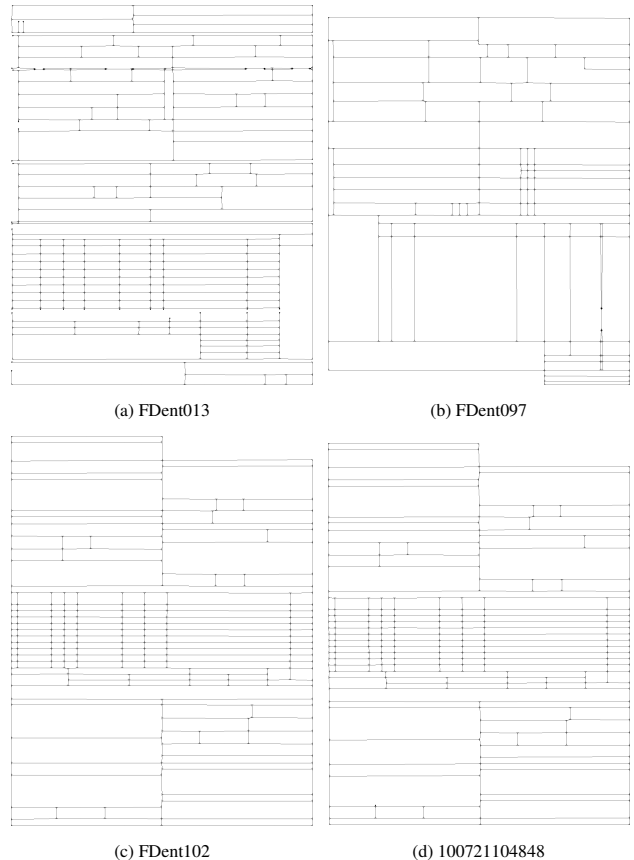


Fig. 4: Examples of American Medical Association (AMA) dental claim forms documents. Among the above ‘FDent013’, ‘FDent097’ and ‘FDent102’ are the three different categories, which are obtained by digitizing and removing the textual parts from the respective blank form templates and ‘100721104848’ is a dental claim form encountered in a production document processing application, which is obtained by digitizing and removing the textual parts from it. This particular form belong to the same class as of ‘FDent102’. (Best viewed in pdf).

The dataset used for this experiment is basically a collection of 6247 American Medical Association (AMA) dental claim forms encountered in a production document processing application. This dataset also includes 208 blank forms which serve as the categories. So the task is to assign each of the 6247 forms one of the labels associated to the 208 categories. In these forms the rectilinear lines intersect each other in well defined ways that form junction and also free end terminator, which essentially serve as the graph nodes and their connections as the graph edges. There are only 13 node labels depending on the junction type (refer to [35] for more details) and only two edge labels: vertical and horizontal.

We have followed the same performance evaluation protocol as [35] for evaluating and comparing our method for

⁷. Available at www2.parc.com/isl/groups/pda/data/DentalFormsLineArtDataSet.zip

this experiment. This particular evaluation scheme is based on comparing the rank ordering of category model matches to the document image graphs between the classifier output and the ground truths. Let $r_{g,c}$ be the ranking assigned by a classifier to the model assigned top rank in the ground truth. Let $r_{c,g}$ be the ranking in the ground truth of the model assigned top rank by the system. Then, the performance measure ρ is defined as

$$\rho = \frac{1}{2} \left(\frac{1}{r_{c,g}} + \frac{1}{r_{g,c}} \right) \quad (1)$$

Under this performance measure a maximum score 1 is only given when the top ranking model categories assigned by the classifier and the ground truth agree. Some credit is also given when the top ranking category of the ground truth or classifier output score highly in the complement rankings. For more details on the performance measure protocol, we refer to [35].

TABLE 7: Performance measure ρ obtained by our method (SGE) for retrieving the AMA dental forms documents into 208 model categories and comparison with the method proposed by Saund [35]. It shows the results varying the size of graphlets and their combination. *hist. int. sim.* refers to feature vector comparison using histogram intersection similarity whereas *cosine sim.* refers to feature vector comparison using cosine similarity. *CMD comp.* refers to feature vector comparison using the CMD distance [35]. *cos comp.* refers to feature vector comparison using the cosine distance. *Extv. G.L. Level* refers to the size of subgraph in terms of number of nodes

SGE			Saund [35]				
Distance or Similarity Measure	Graphlets	Perf. Measure ρ	Graphlets	Perf. Measure ρ	Test Condition	Extv. G.L. Level	Perf. Measure ρ
hist. int. sim.	$t = 0$	0.291	—	—	—	—	—
hist. int. sim.	$t = 1$	0.264	$t = \{0, \dots, 1\}$	0.296	—	—	—
hist. int. sim.	$t = 2$	0.336	$t = \{0, \dots, 2\}$	0.337	—	—	—
hist. int. sim.	$t = 3$	0.382	$t = \{0, \dots, 3\}$	0.390	—	—	—
hist. int. sim.	$t = 4$	0.388	$t = \{0, \dots, 4\}$	0.416	CMD comp.	$\{1, \dots, 2\}$	0.411
hist. int. sim.	$t = 5$	0.393	$t = \{0, \dots, 5\}$	0.435	CMD comp.	$\{1, \dots, 3\}$	0.467
hist. int. sim.	$t = 6$	0.452	$t = \{0, \dots, 6\}$	0.486	CMD comp.	$\{1, \dots, 4\}$	0.507
hist. int. sim.	$t = 7$	0.489	$t = \{0, \dots, 7\}$	0.536	CMD comp.	$\{1, \dots, 5\}$	0.524
cosine sim.	$t = 0$	0.289	—	—	—	—	—
cosine sim.	$t = 1$	0.217	$t = \{0, \dots, 1\}$	0.293	—	—	—
cosine sim.	$t = 2$	0.276	$t = \{0, \dots, 2\}$	0.304	—	—	—
cosine sim.	$t = 3$	0.282	$t = \{0, \dots, 3\}$	0.316	—	—	—
cosine sim.	$t = 4$	0.308	$t = \{0, \dots, 4\}$	0.328	cosine comp.	$\{1, \dots, 2\}$	0.341
cosine sim.	$t = 5$	0.312	$t = \{0, \dots, 5\}$	0.336	cosine comp.	$\{1, \dots, 3\}$	0.353
cosine sim.	$t = 6$	0.323	$t = \{0, \dots, 6\}$	0.361	cosine comp.	$\{1, \dots, 4\}$	0.371
cosine sim.	$t = 7$	0.341	$t = \{0, \dots, 7\}$	0.382	cosine comp.	$\{1, \dots, 5\}$	0.377

For obtaining the performance measure mentioned above, we obtain our proposed stochastic graphlet embedding both for the forms documents and also for the templates with $\epsilon = 0.05$ and $\delta = 0.05$. After obtaining the embeddings, we compute the similarity between each pair of document and template using two different similarity functions *viz.* *histogram intersection* and *cosine* similarity, which respectively correspond to the *Common-Minus-Difference* and *cosine* comparison in [35]. In Table 7, we have shown the performance measures obtained by our stochastic graphlet embedding with graphlets of fixed number of edges and also their combination. Here it is to be mentioned that $t = 0$ corresponds to embedding with graphlets with edge equal to zero *i.e.* only nodes. In this experiment too, it has been seen that increasing number of edges of graphlets have a positive impact on the performance. Furthermore, it

can be observed that combining statistics of graphlets with two or more different number of edges often increases the performance and achieved the best results by our system which essentially surpass the measure obtained by [35].

5.4 MNIST Database

In this section, we show the impact of our proposed stochastic graphlet embedding on the performance of handwritten digit classification task. For that, we consider the well known MNIST database⁸ (see Fig. 5 for example), which consists of 60000 training and 10000 test images belonging to 10 different digit classes. Hence the task of the classifier is to assign each test sample one of the 10 labels, which is a 10-class classification problem.

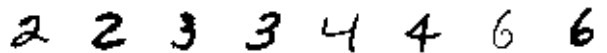


Fig. 5: Sample of image pairs belonging to the same class taken from MNIST.

For obtaining a graph representation of the digits, we skeletonize the binary digit images and consider each pixel on the skeleton as a node, where all the adjacent (considering 8 neighbours) pixels on the skeleton are connected and each connection constitutes an edge. For labelling the graph nodes, we consider the general rotation variant shape context descriptors [3] on each of the nodes and cluster them using k-means clustering algorithm with $k = 20$, which assign each node a numeric label in $[1, 20]$. Given the graph representations of the handwritten digits, we use our stochastic graphlet embedding to obtain the distributions of high order graphlets with $\epsilon = 0.05$ and $\delta = 0.05$, with that we compute a histogram intersection kernel matrix and plug into SVM for training and classification. In practice, we use LIBSVM to train a “one-vs-all” SVM for each handwritten digit class.

Table 8 shows the classification accuracies obtained by our stochastic graphlet embedding, where we present the performance of our method by combining graphlets of different number of edges. It is to be noted that the increasing number of edges has a positive impact on the accuracy of image classification; a clear gain is observed as t increases and this shows that high order graphlets are more discriminating and gradually improve the performance, which is quite encouraging. However, our method fails to obtain the best accuracy on this dataset, which is already very high to be noted.

TABLE 8: Accuracies (in %) obtained by our method with a combination of different graphlet size (t) on the MNIST dataset.

t	$\{1, \dots, 2\}$	$\{1, \dots, 3\}$	$\{1, \dots, 4\}$	$\{1, \dots, 5\}$	$\{1, \dots, 6\}$	$\{1, \dots, 7\}$
Acc.	93.75	95.08	96.15	97.32	98.67	99.20

8. Available at <http://yann.lecun.com/exdb/mnist>

5.5 Kimia Databases

Lastly, we evaluate the performance of our method in shape retrieval problem using the Kimia databases [37]. The first dataset of Kimia contains 216 images belonging of 18 different object classes with many protrusions and rotated objects (see Fig. 6), this database is a subset of the well known MPEG-7 dataset. Kimia’s second dataset consists of 99 images from 9 distinct classes, this database contains objects also having protrusions and missing parts (see Fig. 7).



Fig. 6: Sample of image pairs belonging to the same classes, taken from Kimia’s 216 database.



Fig. 7: Sample of image pairs belonging to the same classes, taken from Kimia’s 99 database.

For obtaining a graph representation of a shape, we consider each pixel on the shape as a node where all the adjacent (considering 8 neighbours) pixels on the shape are connected and each connection constitutes an edge. For labelling the graph nodes, we consider the rotation invariant shape context descriptors [3] on each of the nodes and cluster them using k -means clustering algorithm with $k = 50$, which assign each node a numeric label in $[1, 50]$. Given the graph representations of the binary shapes, we use our stochastic graphlet embedding to obtain the distributions of high order graphlets with $\epsilon = 0.05$ and $\delta = 0.05$.

Given a shape, described with our SGE representation, the goal is to find the set of its k -nearest neighbours. In these experiments, performance evaluation follows a standard protocol, where for each image, we check whether its k closest neighbours (with $k = 11$ for Kimia’s 216 and $k = 10$ for Kimia’s 99 dataset) belong to the same class as the query or not, which produces k binary responses. We follow this procedure for all the images in a dataset and sum up the binary responses for the entire dataset, which act as a performance indicator for a particular method. Table 9 shows our results on the Kimia’s 216 dataset, together with some of the popular state-of-the-art methods. It can be seen that among the existing methods, Shock Graph [37] performed the best followed by Skeleton Graph [2]. However, it is clear that our high order stochastic graphlet embedding is able to discriminate between different classes quite accurately while also achieving rotation invariance and to some extent protrusion invariance, and outperforms the Shape Context [3] and PHOG [27].

Table 10 shows our results and comparisons with the four the state-of-the-art methods on Kimia’s 99 dataset. As of Kimia’s 216 database, on this dataset also our method

TABLE 9: Retrieval results obtained by our method (SGE) on the Kimia’s 216 dataset for different values of k (in k -nn). These results correspond to the number of relevant images returned using k -nn.

Algorithm	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
Shape Context [3]	214	209	205	197	191	178	161	144	131	101	78
Shock Graph [37]	216	216	216	215	210	210	207	204	200	187	163
Skeleton Graph [2]	216	216	215	216	213	210	210	207	205	191	177
PHOG [27]	203	194	188	178	172	170	164	163	151	152	121
SGE $t = \{0, \dots, 6\}$	156	148	146	142	132	125	120	117	111	101	106
SGE $t = \{0, \dots, 7\}$	172	156	152	146	140	132	126	123	120	116	106
SGE $t = \{0, \dots, 8\}$	178	162	158	153	146	140	132	126	122	120	106
SGE $t = \{0, \dots, 9\}$	194	182	174	168	160	155	148	140	132	128	106
SGE $t = \{0, \dots, 10\}$	205	200	192	184	174	168	160	150	138	130	114
SGE $t = \{0, \dots, 11\}$	215	211	204	198	193	178	161	153	138	134	108

performs upto an acceptable level, but not the best. The low performance may be because of the presence of significant deformation, which can be addressed with a different type of graph representation in our case.

TABLE 10: Retrieval results obtained by our proposed method (SGE) on the Kimia’s 99 dataset for different values of k (in k -nn). These results correspond to the number of relevant images returned using k -nn.

Algorithm	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Shape Context [3]	97	91	88	85	84	77	75	66	56	37
Shock Graph [37]	99	99	99	98	98	97	96	95	93	82
Skeleton Graph [2]	99	99	99	99	96	97	95	93	89	73
PHOG [27]	91	83	73	77	67	59	48	54	45	31
SGE $t = \{0, \dots, 6\}$	78	75	71	67	63	55	51	44	41	38
SGE $t = \{0, \dots, 7\}$	81	79	75	69	65	59	55	47	43	39
SGE $t = \{0, \dots, 8\}$	86	84	80	74	68	61	54	48	44	41
SGE $t = \{0, \dots, 9\}$	89	89	83	78	70	65	58	51	45	43
SGE $t = \{0, \dots, 10\}$	95	93	88	84	78	73	69	61	56	51
SGE $t = \{0, \dots, 11\}$	99	98	97	95	91	83	78	74	71	65

6 CONCLUSIONS

In this paper, we have successfully introduced high order stochastic graphlet embedding for graph-based pattern classification problem. For that, we have proposed a stochastic graph parsing algorithm which is based on a stochastic depth first search procedure for sampling large number of connected graphlets with desired number of edges. For obtaining the distribution of these sampled graphlets, it is needed to identify the isomorphic ones which is a GI-complete problem for generic graph. In this work, we have suggested to do that with graph hashing techniques. In order to achieve that, we have designed effective graph hash functions with low probability of collision, which can generate same hash codes for isomorphic graphlets and different hash codes for non-isomorphic ones with a very few exceptions. In summary, our proposed method obtains the distribution of high order graphlets while avoiding systematic generation of huge dictionaries of graphlets and laborious exact graph isomorphism search procedure of these graphlets. We have performed a very detailed and robust experimentation showing the effectiveness of our proposal. It has been corroborated through experimental results that high order graphlets really contain robust information for graph-based pattern recognition. Moreover, our method is versatile and can be applied to any graph-based pattern recognition problem, this fact has also been proven by experiments on diverse datasets.

A further improvement of our method can be done by avoiding the minimal graph hash code collision that still exists in our graph hashing protocol. To achieve that, combining two or more different graph hash functions could be useful. In addition, eliminating one of the colliding graphlets from consideration might be effective. However, all these need further investigations on this topic which will be done as a future work. Moreover, extending the proposed method for graphs with large number of real attributes will also be studied in future.

ACKNOWLEDGMENTS

Anjan Dutta was a postdoctoral researcher at the Télécom ParisTech in Paris when some of the works described in this paper were done. He would like to thank Prof. Josep Lladós for some fruitful discussions related to this topic.

REFERENCES

- [1] F. Aziz, R. Wilson, and E. Hancock, "Backtrackless walks on a graph," *IEEE TNNLS*, vol. 24, no. 6, pp. 977–989, 2013.
- [2] X. Bai and L. J. Latecki, "Path similarity skeleton graph matching," *IEEE TPAMI*, vol. 30, pp. 1282–1292, 2008.
- [3] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE TPAMI*, vol. 24, no. 4, pp. 509–522, 2002.
- [4] K. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *ICDM*, 2005, pp. 74–81.
- [5] H. Bunke and K. Riesen, "Improving vector space embedding of graphs through feature selection algorithms," *PR*, vol. 44, no. 9, pp. 1928 – 1940, 2010.
- [6] T. Caelli and S. Kosinov, "An eigenspace projection clustering method for inexact graph matching," *IEEE TPAMI*, vol. 26, no. 4, pp. 515 –519, 2004.
- [7] M. Cho, J. Lee, and K. Lee, "Reweighted random walks for graph matching," in *ECCV*, 2010, pp. 492–505.
- [8] F. Comellas and J. Paz-Sánchez, "Reconstruction of networks from their betweenness centrality," in *AEC*, 2008, pp. 31–37.
- [9] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *IJPRAI*, vol. 18, no. 3, pp. 265–298, 2004.
- [10] G. Csürka, C. Dance R., L. Fan, J. Williamowski, and C. Bray, "Visual categorization with bags of keypoints," in *SLCVW, ECCV*, 2004, pp. 1–22.
- [11] N. Dahm, H. Bunke, T. Caelli, and Y. Gao, "A unified framework for strengthening topological node features and its application to subgraph isomorphism detection," in *GbrPR*, 2013, pp. 11–20.
- [12] O. Duchenne, A. Joulain, and J. Ponce, "A graph-matching kernel for object categorization," in *ICCV*, 2011, pp. 1792–1799.
- [13] F.-X. Dupé and L. Brun, "Hierarchical bag of paths for kernel based shape classification," in *S+SSPR*, 2010, pp. 227–236.
- [14] P. Foggia, G. Percannella, and M. Vento, "Graph matching and learning in pattern recognition in the last 10 years," *IJPRAI*, vol. 28, no. 1, pp. 1–40, 2014.
- [15] T. Gärtner, "A survey of kernels for structured data," *ACM SIGKDD*, vol. 5, no. 1, pp. 49–58, 2003.
- [16] J. Gibert, E. Valveny, and H. Bunke, "Graph embedding in vector spaces by node attribute statistics," *PR*, vol. 45, no. 9, pp. 3072 – 3083, 2012.
- [17] M. Gori, M. Maggini, and L. Sarti, "Exact and approximate graph matching using random walks," *IEEE TPAMI*, vol. 27, no. 7, pp. 1100–1111, 2005.
- [18] Z. Harchaoui and F. Bach, "Image classification with segmentation graph kernels," in *CVPR*, 2007, pp. 1–8.
- [19] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *KDD*, 2004, pp. 158–167.
- [20] J. Kandola, N. Cristianini, and J. S. Shawe-taylor, "Learning semantic similarity," in *NIPS*, 2002, pp. 673–680.
- [21] J. Kim and K. Grauman, "Asymmetric region-to-image matching for comparing images with generic object categories," in *CVPR*, 2010, pp. 2344–2351.
- [22] J. Lafferty and G. Lebanon, "Diffusion kernels on statistical manifolds," *JMLR*, vol. 6, pp. 129–163, 2005.
- [23] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR*, 2006, pp. 2169–2178.
- [24] W.-J. Lee and R. P. W. Duin, "A labelled graph based multiple classifier system," in *MCS*, 2009, pp. 201–210.
- [25] H. Ling and D. Jacobs, "Shape classification using the inner-distance," *IEEE TPAMI*, vol. 29, no. 2, pp. 286–299, 2007.
- [26] M. M. Luqman, J.-Y. Ramel, J. Lladós, and T. Brouard, "Fuzzy multilevel graph embedding," *PR*, vol. 46, no. 2, pp. 551–565, 2013.
- [27] S. Maji and J. Malik, "Fast and accurate digit classification," UoC, Berkeley, Tech. Rep., 2009.
- [28] K. Mehlhorn, *Graph algorithms and NP-completeness*, 1984.
- [29] S. F. Mousavi, M. Safayani, A. Mirzaei, and H. Bahonar, "Hierarchical graph embedding in vector space by graph pyramid," *PR*, vol. 61, pp. 245 – 254, 2017.
- [30] M. Neuhaus and H. Bunke, *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific, 2007.
- [31] M. J. Newman, "A measure of betweenness centrality based on random walks," *SN*, vol. 27, no. 1, pp. 39 – 54, 2005.
- [32] E. Pekalska and R. P. W. Duin, *The Dissimilarity Representation for Pattern Recognition: Foundations And Applications*. World Scientific, USA, 2005.
- [33] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *S+SSPR*, 2008, pp. 287–297.
- [34] A. Robles-Kelly and E. R. Hancock, "A riemannian approach to graph embedding," *PR*, vol. 40, no. 3, pp. 1042 – 1056, 2007.
- [35] E. Saund, "A graph lattice approach to maintaining and learning dense collections of subgraphs as image features," *IEEE TPAMI*, vol. 35, no. 10, pp. 2323–2339, 2013.
- [36] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.
- [37] T. Sebastian, P. Klein, and B. Kimia, "Recognition of shapes by editing their shock graphs," *IEEE TPAMI*, vol. 26, no. 5, pp. 550–571, 2004.
- [38] A. Sharma, R. Horaud, J. Cech, and E. Boyer, "Topologically-robust 3d shape matching based on diffusion geometry and seed growing," in *CVPR*, 2011, pp. 2481–2488.
- [39] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *AISTATS*, 2009, pp. 488–495.
- [40] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in *NIPS*, 2009, pp. 1660–1668.
- [41] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *COLT*, 2003, pp. 144–158.
- [42] V. N. Vapnik, *Statistical Learning Theory*. Wiley, 1998.
- [43] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *JMLR*, vol. 11, pp. 1201–1242, 2010.
- [44] C. Watkins, "Kernels from matching operations," University of London, Computer Science Department, Tech. Rep., 1999.
- [45] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu, and M. J. Weinberger, "Inequalities for the H deviation of the empirical distribution," HP Labs, Palo Alto, Tech. Rep., 2003.
- [46] R. Wilson, E. Hancock, and B. Luo, "Pattern vectors from algebraic graph theory," *IEEE TPAMI*, vol. 27, no. 7, pp. 1112 –1124, 2005.
- [47] B. Wu, C. Yuan, and W. Hu, "Human action recognition based on context-dependent graph kernels," in *CVPR*, 2014, pp. 2609–2616.
- [48] F. Zhou and F. De la Torre, "Deformable graph matching," in *CVPR*, 2013, pp. 1–8.



Anjan Dutta obtained PhD in Computer Science from the Universitat Autònoma de Barcelona (UAB) in the year 2014. He is a recipient of the Extraordinary PhD Thesis Award for the year 2013-14 by the UAB. Before his PhD, he obtained MS in Computer Vision and Artificial Intelligence also from the UAB, MCA in Computer Applications from the West Bengal University of Technology and BS in Mathematics (Honors) from the University of Calcutta respectively in the year 2010, 2009 and 2006. His recent research interests have revolved around graph-based representation for visual objects and graph-based algorithms for solving various tasks in Computer Vision, Pattern Recognition and Machine Learning.