Estimating Descriptors for Large Graphs

Zohair Raza Hassan[1], Mudassir Shabbir[1], Imdadullah Khan[2(✉)], and Waseem Abbas[3]

[1] Information Technology University of the Punjab, Lahore, Pakistan
{zohair.raza,mudassir.shabbir}@itu.edu.pk
[2] Lahore University of Management Sciences, Lahore, Pakistan
imdad.khan@lums.edu.pk
[3] Vanderbilt University, Nashville, USA
waseem.abbas@vanderbilt.edu

Abstract. Embedding networks into a fixed dimensional feature space, while preserving its essential structural properties is a fundamental task in graph analytics. These feature vectors (graph descriptors) are used to measure the pairwise similarity between graphs. This enables applying data mining algorithms (e.g classification, clustering, or anomaly detection) on graph-structured data which have numerous applications in multiple domains. State-of-the-art algorithms for computing descriptors require the entire graph to be in memory, entailing a huge memory footprint, and thus do not scale well to increasing sizes of real-world networks. In this work, we propose streaming algorithms to efficiently approximate descriptors by estimating counts of sub-graphs of order $k \leq 4$, and thereby devise extensions of two existing graph comparison paradigms: the Graphlet Kernel and NetSimile. Our algorithms require a single scan over the edge stream, have space complexity that is a fraction of the input size, and approximate embeddings via a simple sampling scheme. Our design exploits the trade-off between available memory and estimation accuracy to provide a method that works well for limited memory requirements. We perform extensive experiments on real-world networks and demonstrate that our algorithms scale well to massive graphs.

Keywords: Graph descriptor · Edge stream · Graph classification

1 Introduction

Evaluating similarity or distance between a pair of graphs is a building block of many fundamental data analysis tasks on graphs such as classification and clustering. These tasks have numerous applications in social network analysis,

The first two authors have been supported by the grant received to establish CIPL and the third author has been supported the grant received to establish SEIL, both associated with the National Center in Big Data and Cloud Computing, funded by the Planning Commission of Pakistan.

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12084, pp. 779–791, 2020.
https://doi.org/10.1007/978-3-030-47426-3_60

bioinformatics, computational chemistry, and graph theory in general. Unfortunately, large orders (number of vertices) and massive sizes (number of edges) prove to be challenging when applying general-purpose data mining techniques on graphs. Moreover, in many real-world scenarios, graphs in a dataset have varying orders and sizes, hindering the application of data mining algorithms devised for vector spaces. Thus, devising a framework to compare graphs with different orders and sizes would allow for rich analysis and knowledge discovery in many practical domains.

However, graph comparison is a difficult task; the best-known solution for determining whether two graphs are structurally the same takes quasi-polynomial time [1], and determining the minimum number of steps to convert one graph to another is NP-Hard [16]. In a more practical approach, graphs are first mapped into fixed dimensional feature vectors, where vector space-based algorithms are then employed. In a supervised setting, these feature vectors are learned through neural networks [14,25,26]. In unsupervised settings, the feature vectors are descriptive statistics of the graph such as average degree, the eigenspectrum, or spectra of sub-graphs of order at most k contained in the graph [7,11,17,18,22,23].

The runtimes and memory costs of these methods depend directly on the magnitude (order and size) of the graphs and the dimensionality (dependent on the number of statistics) of the feature-space. While computing a larger number of statistics would result in richer representations, these algorithms do not scale well to the increasing magnitudes of a real-world graphs [9].

A promising approach is to process graphs as streams - one edge at a time, without storing the whole graph in memory. In this setting, the graph descriptors are approximated from a representative sample achieving practical time and space complexity [6,15,19–21].

In this work we propose GABE (Graphlet Amounts via Budgeted Estimates), and MAEVE (Moments of Attributes Estimated on Vertices Efficiently), stream-based extensions of the Graphlet Kernel [17], and NetSimile [3], respectively. Our contributions can be summarised as follows:

- We propose two simple and intuitive descriptors for graph comparisons that run in the streaming setting.
- We provide analytical bounds on the time and space complexity of our feature vectors generation; for a fixed budget, the runtime and space cost of our algorithms are linear.
- We perform extensive empirical analysis on benchmark graph classification datasets of varying magnitudes. We demonstrate that GABE and MAEVE are comparable to the state-of-the-art in terms of classification accuracy, and scale to networks with millions of nodes and edges.

The rest of the paper is organized as follows. We discuss the related work in Sect. 2. Section 3 discusses all preliminaries required to read the text. We present GABE and MAEVE in Sect. 4. We report our experimental findings in Sect. 5 and finally conclude the paper in Sect. 6.

2 Related Work

Methods for comparing a pair of graphs can broadly be categorized into *direct approaches*, *kernel methods*, *descriptors*, and *neural models*. Direct approaches for evaluating the similarity/distance between a pair of graphs preserve the entire structure of both graphs. The most prominent method under this approach is the *Graph Edit Distance* (GED), which counts the number of edit operations (insertion/deletion of vertices/edges) required to convert a given graph to another [16]. Although intuitive, GED is stymied by its computational intractability. Computing distance based on the vertex permutation that minimizes the "error" between the adjacency representations of two graphs is a difficult task [1], and proposed relaxations of these distances are not robust to permutation [2]. An efficient algorithm for large network comparison DeltaCon, is proposed in [9] but it is only feasible when there is a valid one-to-one correspondence between vertices of the two graphs.

In the kernel-based approach, graphs are mapped to a fixed dimensional vector space based on various substructures in the graphs. A kernel function is then defined, which serves as a pairwise similarity measure that takes as input a pair of graphs and outputs a non-negative real number. Typically, the kernel value is the inner-product between two feature vectors corresponding to the two graphs. This so-called kernel trick has been used successfully to evaluate pairwise of other structures such as images and sequences [4,10,12]. Several graph kernels based on sub-structural patterns have been proposed, such as the Shortest-Path [5] and Graphlet [17] kernels. More recently, a hierarchical kernel based on propagating spectral information within the graph [11] was introduced. The WL-Kernel [18] that is based on the Weisfeller-Lehman isomorphism test has been shown to provide excellent results for classification and is used as a benchmark in the graph representation learning literature. Kernels require expensive computation and typically necessitate storing the adjacency matrices, making them infeasible for massive graphs.

Graph Neural Networks (GNNs) learn graph level embeddings by aggregating node representations learned via convolving neighborhood information throughout the neural network's layers. This idea has been the basis of many popular neural networks and is as powerful as WL-Kernels for classification [14,26]. We refer interested readers to a comprehensive survey of these models [25]. Unfortunately, these models also require expensive computation and storing large matrices, hindering scalability to real-world graphs.

Graph descriptors, like the above two paradigms, attempt to map graphs to a vector space such that similar graphs are mapped to closely in the Euclidean space. Generally, the dimensionality of these vectors is small, allowing efficient algorithms for graph embeddings. NetSimile [3] describes graphs by computing moments of vertex features, while SGE [7] uses random walks and hashing to capture the presence of different sub-structures in a graph. State of the art descriptors are based on spectral information; [23] proposed a family of graph spectral distances and embedding the information as histograms on the multiset of distances in a graph, and NetLSD [22] computes the heat (or wave) trace over the eigenvalues of a graph's normalized Laplacian to construct embeddings.

The fundamental limitation of all the above approaches is the requirement that the entire graph is available in memory. This limits the applicability of the methods to a graph of small magnitude. To the best of our knowledge, this work is the first graph comparison method that does not assume this.

Streaming algorithms assume an online setting; the input is streamed one element at a time, and the amount of space we are allowed is limited. This allows one to design scalable approximation algorithms to solve the underlying problems. There has been extensive work on estimating triangles (cycles of length three) in graphs [19,21], butterflies (cycles of length four) in bipartite graphs [15], and anomaly detection [8] when the graph is input as a stream of edges. A framework for estimating the number of connected induced sub-graphs on three and four vertices is presented in [6].

3 Preliminaries and Problem Definition

3.1 Notation and Terminology

Let $G = (V_G, E_G)$ be an undirected, unweighted, simple graph, where V_G is the set of vertices and E_G is the set of edges.

For $v \in V_G$, let $N_G(v) = \{u : (u, v) \in E_G\}$ be the set of neighbors of v, and $d_G^v := |N_G(v)|$ the degree of v. A graph is connected if and only if there exists a path between all pairs in V_G.

A sub-graph of G is a graph, $G' = (V_{G'}, E_{G'})$, such that $V_{G'} \subseteq V_G$ and $E_{G'}$ is a subset of edges in E_G that are incident only on the vertices present in $V_{G'}$, i.e. $E_{G'} \subseteq \{(u, v) : (u, v) \in E_G \wedge u, v \in V_{G'}\}$. If equality holds ($E_{G'}$ contains all edges from the original graph), then G' is called an induced sub-graph of G.

Two graphs, G_1 and G_2, are isomorphic if and only if there exists a permutation $\pi : V_{G_2} \rightarrow V_{G_1}$ such that $E_{G_1} = \{(\pi(u), \pi(v)) : (u, v) \in E_{G_2}\}$. For a graph $F = (V_F, E_F)$, let H_G^F (resp. $\widehat{H}_G^F$) be the set of sub-graphs (resp. induced sub-graphs) of G that are isomorphic to F.

We assume vertices in V_G are denoted by integers in the range $[0, |V_G| - 1]$. Let $S = e_1, e_2, \ldots, e_{|E_G|}$ be a sequence of edges in an arbitrary but fixed order, i.e. $e_t = (u_t, v_t)$ is the t^{th} edge. Let b be the maximum number of edges (budget) one can store in our sample, referred to as $\widetilde{E_G}$.

3.2 Problem Definition

We now formally define the graph descriptor problem:

Problem 1 (Constructing Graph Descriptors). Let $\mathcal{G}$ be the set of all possible undirected, unweighted, simple graphs. We wish to find a function, $\varphi : \mathcal{G} \rightarrow \mathbb{R}^d$, that can map any given graph to a d-dimensional vector.

Existing work [3,22] on graph descriptors asserts that the underlying algorithms should be able to run on any graph, regardless of order or size, and should output the same representation for different vertex permutations. Moreover, the descriptors should capture features that can be compared across graphs of

varying orders; directly comparing sub-graph counts is illogical as bigger graphs will naturally have more sub-graphs. The descriptors we propose are based on graph comparison methods that meet these requirements due to their graph-theoretic nature and feature scaling based on the graph's magnitude. We consider an online setting and model the input graph as a stream of edges. We impose the following constraints on our algorithms:

C1: Single Pass: The algorithm is only allowed to receive the stream once.
C2: Limited Space: The algorithm can store a maximum of b edges at once.
C3: Linear Complexity: Space and time complexity of the algorithms should be linear (for fixed b) with respect to the order and size of the graph.

3.3 Estimating Connected Sub-graph Counts on Streams

Problem 2 (Connected Sub-graph Estimation on Streams). Let S be a stream of edges, $e_1, e_2, \ldots, e_{|E_G|}$ for some graph $G = (V_G, E_G)$. Let $F = (V_F, E_F)$ be a small connected graph such that $|V_F| \ll |V_G|$. Produce an estimate, N_G^F, of $|H_G^F|$ while storing a maximum of b edges at any given instant.

Based on previous works on sub-graph estimation [6,19–21] the underlying recipe for algorithms that solve Problem 2 consists of the following steps:

- For each edge $e_t \in S$, counting the instances of F incident on e_t. For example, if F is a triangle, then it amounts to counting the number of triangles an edge e_t is part of.
- A sampling scheme through which we can compute the probability of detecting F in our sample, denoted by p_t^F, at the arrival of the t^{th} edge.

At the arrival of e_t, we increment our estimate of $|H_G^F|$ by $1/p_t^F$ for all instances of F in our sample $\widetilde{E_G}$ that e_t belongs to. The pseudocode is provided in Algorithm 1. This simple methodology allows one to compute estimates whose expected values are equal to $|H_G^F|$:

Theorem 1. *Algorithm 1 provides unbiased estimates:* $\mathbb{E}[N_G^F] = |H_G^F|$.

Proof. For a sub-graph $h \in H_F^G$, let X_h be a random variable such that $X_h = 1/p_t^F$ if h is detected at the arrival of its last edge in the stream e_t, and 0 otherwise. Clearly, $N_G^F = \sum_{h \in H_G^F} X_h$, and $\mathbb{E}[X_h] = (1/p_t^F) \times p_t^F = 1$. Therefore,

$$\mathbb{E}\left[N_G^F\right] = \mathbb{E}\left[\sum_{h \in H_G^F} X_h\right] = \sum_{h \in H_G^F} \mathbb{E}\left[X_h\right] = \sum_{h \in H_G^F} 1 = \left|H_G^F\right|.$$

At the arrival of e_t, counting only the sub-graphs that e_t belongs to ensures that we do count the same sub-graph twice. In this work, we employ reservoir sampling [24], which has been shown to be effective for sub-graph estimation [6, 20,21]. Using reservoir sampling, the probability of detecting an F that e_t belongs

Algorithm 1: Sub-graph Estimation on Streams

Input : Stream of edges $S = e_1, e_2, \ldots, e_{|E_G|}$, budget b, and a graph F
Output: N_G^F (estimate of $|H_G^F|$)
$\widetilde{E_G} \leftarrow \emptyset$, $N_G^F \leftarrow 0$ `/* Initialize sample of edges, and estimate */`
for $e_t \in S$ **do**
  Find all instances of F that e_t belongs to in $\widetilde{E_G} \cup \{e_t\}$
  Increment N_G^F by $1/p_t^F$ for each F detected
  Sample e_t in $\widetilde{E_G}$, based on b
end

to at the arrival of e_t is equivalent to the probability that $|E_F|-1$ particular edges are present in the sample after $t-1$ time-steps: $p_t^F = \min\left(1, \prod_{i=0}^{|E_F|-2} \frac{b-i}{t-1-i}\right)$.

We now derive an upper bound for the variance. Note that while the bound is loose, it is sufficient to show that we obtain better results with greater b, and applies to any connected graph, F.

Theorem 2. *When using reservoir sampling, the variance of N_G^F in Algorithm 1 is bounded as follows:* $\mathrm{Var}[N_G^F] \leq |H_G^F|^2 \prod_{i=0}^{|E_F|-2} \frac{|E_G|-i}{b-i}$.

Proof. The theorem is trivially true when $b \geq |E_G| - 1$. We now explore the case when $b < |E_G| - 1$. Let X_h be a random variable as defined in the proof for Theorem 1. Note that $p_t^F \geq p_{|E_G|}^F$, and $\mathrm{Var}[X_h] = \mathbb{E}[X_h^2] - \mathbb{E}[X_h]^2 = 1/p_t^F - 1 \leq 1/p_{|E_G|}^F$. We bound the total variance using the Cauchy-Schwarz inequality:

$$\mathrm{Var}[N_G^F] = \sum_{h \in H_G^F} \sum_{h' \in H_G^F} \mathrm{Cov}[X_h, X_{h'}] \leq \sum_{h \in H_G^F} \sum_{h' \in H_G^F} \sqrt{\mathrm{Var}[X_h]\mathrm{Var}[X_{h'}]}$$

$$\leq \sum_{h \in H_G^F} \sum_{h' \in H_G^F} \frac{1}{p_{|E_G|}^F} = |H_G^F|^2 \prod_{i=0}^{|E_F|-2} \frac{|E_G|-1-i}{b-i}.$$

Note that this methodology is also applicable for estimating the number of sub-graphs that each vertex is incident in, and simple modifications to the proofs for Theorems 1 and 2 will prove the same results for estimations on vertex counts.

4 GABE and MAEVE

In this section discuss our two proposed descriptors: Graphlet Amounts via Budgeted Estimates (GABE), which is based on the Graphlet Kernel, and Moments of Attributes Estimated on Vertices Efficiently (MAEVE), based on NetSimile.

4.1 Graphlet Amounts via Budgeted Estimates

Let $\mathcal{F}_k$ be the set of graphs with order k. For two given graphs, G_1 and G_2, Shervashidze et al. [17] propose counting all graphlets (induced sub-graphs) of

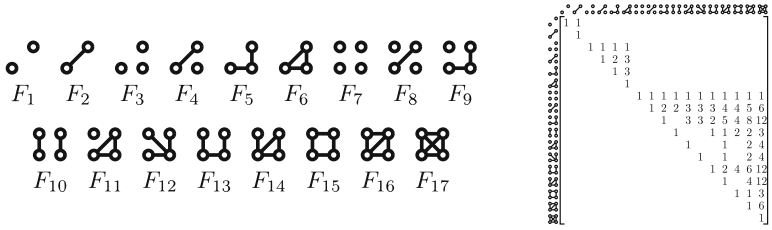


Fig. 1. The graphs counted by GABE, and their corresponding overlap matrix $\mathcal{O}$ (best viewed when zoomed in). Zeros have been omitted for readability.

order k in both graphs, and computing similarity based on the inner product $\langle \phi_k(G_1), \phi_k(G_2) \rangle$, where, for a given k, and graphs $F_i \in \mathcal{F}_k$:

$$\phi_k(G) := \frac{1}{\binom{|V_G|}{k}} \left[\left| \widehat{H}_G^{F_1} \right| \left| \widehat{H}_G^{F_2} \right| \left| \widehat{H}_G^{F_3} \right| \cdots \left| \widehat{H}_G^{F_{|\mathcal{F}_k|-1}} \right| \left| \widehat{H}_G^{F_{|\mathcal{F}_k|}} \right| \right]^\intercal$$

Their algorithm runs in $O(|V_G| d^{k-1})$ ($d = \max_{v \in V_G} d_G^v$) for $k \in \{3, 4, 5\}$, and uses adjacency matrices. We use the methodology of [6], to estimate the sub-graph counts as in Sect. 3.3, then compute induced sub-graph counts based on the overlap of graphs of the same order. We follow this procedure for estimating sub-graph counts of order $k \in \{2, 3, 4\}$, then concatenate the resultant $\phi_k(G)$'s into a vector. The 17 graphs we enumerate are shown in Fig. 1. Note that unlike [6], we also estimate the counts of disconnected induced sub-graphs.

Induced Sub-graph Counts. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_{17}\}$ be the set of graphs we enumerate. Let $\mathcal{H}_G^{\mathcal{F}}$ (resp. $\widehat{\mathcal{H}}_G^{\mathcal{F}}$) be a vector such that i^{th} entry corresponds to $|H_G^{F_i}|$ (resp. $|\widehat{H}_G^{F_i}|$). Let $\mathcal{O}$ be a $|\mathcal{F}| \times |\mathcal{F}|$ matrix such that $O(i, j)$ is the number of sub-graphs of F_j, isomorphic to F_i, when $|V_{F_i}| = |V_{F_j}|$, and 0 otherwise. One can clearly see that $\mathcal{H}_G^{\mathcal{F}} = \mathcal{O}\widehat{\mathcal{H}}_G^{\mathcal{F}}$, as we account for the sub-graph counts that are disregarded when only considering induced sub-graphs. Since $\mathcal{O}$ is an upper triangular matrix, it is invertible. Thereby, given $\mathcal{H}_G^{\mathcal{F}}$, one can retrieve the induced sub-graph counts by computing $\mathcal{O}^{-1}\mathcal{H}_G^{\mathcal{F}}$. By linearity of expectation, Theorem 1 implies that the induced sub-graph counts are unbiased as well.

While processing the stream, we store the degree of each vertex, by incrementing the degree for u_t, v_t when $e_t = (u_t, v_t)$ arrives. We use edge-centric algorithms (as described in Sect. 3.3) to compute estimates for $F_6, F_{13}, \ldots, F_{17}$, and use intuitive combinatorial formulas, listed in Table 1, to compute the remaining 11 sub-graphs. We can compute $|E_G|$ and $|V_G|$ by keeping track of how many edges have been received, and the maximum vertex label received, respectively.

Time and Space Complexity. An array of size $|V_G|$ is used to store degrees, which can be accessed in $O(1)$ time, and hence the counts for F_5 and F_{12} can be incremented each time an edge arrives in $O(1)$. Let G' denote the graph represented by $\widetilde{E_G}$, stored as an adjacency list. Determining if two vertices are adjacent takes $O(\log b)$ time when using a tree data-structure within the stored

Table 1. Graphs and their corresponding sub-graph count formulas.

Graph	Formula	Graph	Formula	Graph	Formula
	$\binom{\lvert V_G\rvert}{2}$		$\binom{\lvert V_G\rvert}{3}$		$\binom{\lvert V_G\rvert}{4}$
	$\lvert E_G\rvert$		$\lvert E_G\rvert(\lvert V_G\rvert-2)$		$\lvert E_G\rvert\binom{\lvert V_G\rvert-2}{2}$
	$\binom{\lvert E_G\rvert}{2}-\lvert H_G^{F_5}\rvert$		$\sum_{v\in V_G}\binom{d_G^v}{2}$		$\sum_{v\in V_G}\binom{d_G^v}{3}$
	$\lvert H_G^{F_5}\rvert(\lvert V_G\rvert-3)$		$\lvert H_G^{F_6}\rvert(\lvert V_G\rvert-3)$	-	-

Table 2. Features extracted for each vertex, $v \in V_G$ for MAEVE, their formulae, and a figure highlighting the relevant edges. The filled in vertex depicts v.

Degree	Clustering Coefficient	Avg. Degree of $N_G(v)$	Edges in $I_G(v)$	Edges leaving $I_G(v)$
d_G^v	$\lvert T_G(v)\rvert/\binom{d_G^v}{2}$	$1+\lvert P_G(v)\rvert/d_G^v$	$d_G^v+\lvert T_G(v)\rvert$	$\lvert P_G(v)\rvert-2\lvert T_G(v)\rvert$
				

adjacency list. At the arrival of $e_t = (u_t, v_t)$, we need to visit only the vertices two hops away from u_t (resp. v_t), then perform at most three adjacency checks. Thereby, we perform $2\left(\sum_{w\in N_{G'}(u_t)} d_{G'}^w + \sum_{w\in N_{G'}(v_t)} d_{G'}^w\right)\times 3\log b = O(b\log b)$ operations for one edge, and $O(b\log b|E_G|)$ in total. Storing an adjacency list with b edges, and an array for degrees takes $O(b+|V_G|)$ space.

4.2 Moments of Attributes Estimated on Vertices Efficiently

NetSimile [3] propose extracting features for each vertex and aggregating them by taking moments over their distribution. Similarly, we propose extracting a subset of those features, listed in Table 2, and computing four moments for each feature: mean, standard deviation, skewness, and kurtosis.

Extracting Vertex Features. For a graph G, and a vertex $v \in V_G$, we use $I_G(v)$ to denote the induced sub-graph of G formed by v and its neighbors. Let $T_G(v)$ be the set of triangles that v belongs to, and $P_G(v)$ be the set of three-paths (paths on three vertices) where v is an end-point. We compute the features in Table 2 by using their formulas on estimates of $|T_G(v)|$, $|P_G(v)|$, and d_G^v computed for each $v \in V$ as in Sects. 3.3 and 4.1.

Theorem 3. *For a vertex $v \in V_G$, all vertex features used in* MAEVE *can be expressed in terms of d_G^v, $|T_G(v)|$, and $|P_G(v)|$.*

Proof. The first two are already expressed in terms of d_G^v and $|T_G(v)|$.

Average Degree of Neighbors: For each $u \in N_G(v)$, there is exactly one edge connected to v, accounting for d_G^v edges. The remaining edges are part of three-paths on which v is an end-point. Therefore, $\sum_{u\in N_G(v)} d_G^u = d_G^v + |P_G(v)|$.

Edges in $I_G(v)$: There are two types of edges in $E_{I_G(v)}$: (1) edges incident on v, of which there are d_G^v, and (2) edges not incident on v. The latter must belong to a pair of vertices which form a triangle with v. For each such edge, there is exactly one triangle. Therefore, $\left|E_{I_G(v)}\right| = d_G^v + |T_G(v)|$.

Edges leaving $I_G(v)$: Consider a sub-graph $h \in P_G(v)$. Let u be the other end-point of h, and w be the center vertex. When $u \notin N_G(v)$, it belongs to a three-path that is not in $N_G(v)$, and is thereby an edge leaving the induced sub-graph of v. Now, consider $u \in N_G(v)$. Clearly, the edge (u, w) forms a triangle, and is incident in exactly two three-paths: $\{(v, u), (u, w)\}$ and $\{(v, w), (u, w)\}$. Therefore, if we account for the three-paths that lie within $N_G(v)$, we can formulate the number of edges leaving $I_G(v)$ as $|P_G(v)| - 2|T_G(v)|$.

Time and Space Complexity. As in Sect. 4.1, we assume an adjacency list with an underlying tree structure and refer to the sampled graph as G'. At the arrival of an edge $e_t = (u_t, v_t)$, one can traverse the neighborhoods to obtain the triangle and three-path count in $(N_{G'}(u_t) + N_{G'}(v_t)) + N_{G'}(u_t) + N_{G'}(v_t) = O(b)$ time. We store three arrays of size $|V_G|$ to store degrees, triangle counts, and three-path counts. We can compute the moments in at most two passes over these arrays, giving us a total of $O(b|E_G| + |V_G|)$ time. Storing an adjacency list of size b and arrays of size $|V_G|$ gives us $O(b + |V_G|)$ space.

Improving Estimation Quality with Multiple Workers. Multiple worker machines can be used in parallel to independently estimate triangle counts before aggregating them [20]. Using W worker machines decreases the variances by a factor of $1/W$. Their methodology can be adopted *mutatis mutandis* in our algorithms to improve the estimation quality.

5 Experimental Evaluation

In this section, we perform experiments to show how the approximation quality changes with respect to b, explore how the descriptors perform on classification tasks, and showcase the scalability of the algorithms. As in [3], from extensive experiments, we found that Canberra distance $\left(d(\boldsymbol{x}, \boldsymbol{y}) := \sum_{i=1}^{d} \frac{|\boldsymbol{x}_i - \boldsymbol{y}_i|}{|\boldsymbol{x}_i| + |\boldsymbol{y}_i|}\right)$ performs best when comparing the descriptors. We refer to the approximation error as the distance between the true vectors and their approximations.

Implementation. All experiments were performed on a machine with 48 Intel Xeon E5-2680 v3 @ 2.50GHz Processors, and 125 GB RAM. The algorithms are implemented[1] in C++ using MPICH 3.2 on the base code provided by the authors of Tri-Fly [20]. We use 25 processes to simulate 1 Master and 24 workers. Each descriptor is computed exactly once under this setting.

Datasets. We evaluate our models on randomly sampled REDDIT graphs[2], five benchmark classification datasets with large graphs: D&D, COLLAB, REDDIT-BINARY, REDDIT-MULTI-5K, and REDDIT-MULTI-12K [27] (Table 3), and

[1] Code: https://github.com/zohair-raza/estimating-graph-descriptors/.
[2] https://dynamics.cs.washington.edu/data.html.

Table 3. Details of classification datasets. The number of graphs, classes, and minimum/maximum number of vertices/edges in a graph have been provided.

Dataset	$\lvert\mathcal{G}\rvert$	Classes	$\max \lvert V_G \rvert$	$\max \lvert E_G \rvert$
D&D	1,178	2	5,748	14,267
COLLAB	5,000	3	492	40,120
REDDIT-BINARY	2,000	2	3,782	4,071
REDDIT-MULTI-5K	4,999	5	3,648	4,783
REDDIT-MULTI-12K	11,929	11	3,782	5,171

Table 4. Massive networks with their order, size, and what they represent.

Graph	$\lvert V_G \rvert$	$\lvert E_G \rvert$	Network type
Patent	3,774,768	16,518,937	Citation
Flickr	2,302,925	22,838,276	Friendship
Full USA	23,947,347	28,854,312	Road
UK Domain 2002	18,483,186	261,787,258	Hyperlink

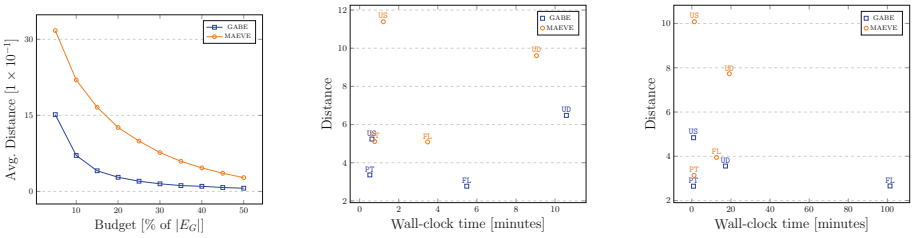
Table 5. Classification accuracy on the datasets described in Table 3. Results within 1% of the best have been boldfaced.

Descriptor	DD	COLLAB	RDT-2	RDT-5	RDT-12
NetLSD [22]	**70.36%**	**74.27%**	82.85%	**41.23%**	30.9%
GABE ($b = 1/4\lvert E_G \rvert$)	65.23%	63.62%	84.65%	**41.1%**	32.18%
GABE ($b = 1/2\lvert E_G \rvert$)	69.08%	65.23%	**85.35%**	40.63%	**32.96%**
MAEVE ($b = 1/4\lvert E_G \rvert$)	59.44%	68.42%	85.04%	41.15%	32.57%
MAEVE ($b = 1/2\lvert E_G \rvert$)	61.26%	70.95%	**86.15%**	**41.53%**	**33.69%**

massive networks from KONECT [13] (Table 4). For each graph, we remove duplicated edges and self-loops, convert to edge-list format with vertex labels in the range $[0, |V_G| - 1]$, and randomly shuffle the list.

5.1 Approximation Quality

We uniformly sampled 1000 graphs of size 10,000 to 50,000 from REDDIT, representing interactions in various "sub-reddits". In Fig. 2(a) we show how the average approximation error taken over all the sampled graphs decreases as b (a fraction of the number of edges) increases.



(a) Error vs. b (b) $b = 100,000$ (c) $b = 500,000$

Fig. 2. Approximation error and runtime of GABE and MAEVE (best viewed in color).

5.2 Graph Classification

We computed descriptors for graphs in Table 3 from samples of 25% and 50% of all the edges and examined their classification accuracy. We used the state-of-the-art descriptor, NetLSD [22], as a benchmark, despite the fact that our models have no direct competitors. As in [22], we used a simple 1-Nearest Neighbour classifier. We performed 10-fold cross-validation for 10 different random splits of the dataset (i.e. 100 different folds are tested on), and report the average accuracy in Table 5. Note that despite using only a fraction of edges, GABE and MAEVE give results competitive to the state of the art.

5.3 Scaling to Large Real-World Networks

We run our algorithms on massive networks (Table 4) and estimated descriptors by setting b to 100,000 and 500,000. In Figs. 2(b) and (c), we show the scatter plots for wall-clock time taken vs. the distance between the real vectors and their approximations (values nearer to the origin are better). We are able to process a graph with ≈260 million edges under 20 min, with relatively low error. Note that when $b = 500,000$, GABE takes 102 min to compute the descriptor for Flickr, implying that we must take the density of the graph into account for efficient computation when setting the value of b.

6 Conclusion

In this work, we present single-pass streaming algorithms to construct graph descriptors using a fixed amount of memory. We show that these descriptors provide better approximations with increasing b, are comparable with the state-of-the-art known descriptors in terms of classification accuracy, and scale well to networks with millions of vertices and edges.

References

1. Babai, L.: Graph isomorphism in quasipolynomial time. In: STOC, pp. 684–697 (2016)
2. Bento, J., Ioannidis, S.: A family of tractable graph distances. In: SDM, pp. 333–341 (2018)
3. Berlingerio, M., Koutra, D., Eliassi-Rad, T., Faloutsos, C.: Network similarity via multiple social theories. In: ASONAM, pp. 1439–1440 (2013)
4. Bo, L., Ren, X., Fox, D.: Kernel descriptors for visual recognition. In: NIPS, pp. 244–252 (2010)
5. Borgwardt, K., Kriegel, H.: Shortest-path kernels on graphs. In: ICDM, pp. 74–81 (2005)
6. Chen, X., Lui, J.: A unified framework to estimate global and local graphlet counts for streaming graphs. In: ASONAM, pp. 131–138 (2017)
7. Dutta, A., Sahbi, H.: Stochastic graphlet embedding. IEEE Trans. Neural Netw. Learn. Syst. **30**(8), 2369–2382 (2019)
8. Eswaran, D., Faloutsos, C.: SedanSpot: detecting anomalies in edge streams. In: ICDM, pp. 953–958 (2018)
9. Faloutsos, C., Koutra, D., Vogelstein, J.: DeltaCon: a principled massive-graph similarity function. In: SDM, pp. 162–170 (2013)
10. Farhan, M., Tariq, J., Zaman, A., Shabbir, M., Khan, I.: Efficient approximation algorithms for strings kernel based sequence classification. In: NIPS, pp. 6935–6945 (2017)
11. Kondor, R., Pan, H.: The multiscale laplacian graph kernel. In: NeurIPS, pp. 2982–2990 (2016)
12. Kuksa, P., Khan, I., Pavlovic, V.: Generalized similarity kernels for efficient sequence classification. In: SDM, pp. 873–882 (2012)
13. Kunegis, J.: KONECT: the Koblenz network collection. In: WWW, pp. 1343–1350 (2013)
14. Morris, C., et al.: Weisfeiler and Leman go neural: higher-order graph neural networks. In: AAAI, pp. 4602–4609 (2019)
15. Sanei-Mehri, S., Zhang, Y., Sariyüce, A.E., Tirthapura, S.: FLEET: butterfly estimation from a bipartite graph stream. In: CIKM, pp. 1201–1210 (2019)
16. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. **13**(3), 353–362 (1983)
17. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: AISTATS, pp. 488–495 (2009)
18. Shervashidze, N., et al.: Weisfeiler-Lehman graph kernels. J. Mach. Learn. Res. **12**, 2539–2561 (2011)
19. Shin, K.: WRS: waiting room sampling for accurate triangle counting in real graph streams. In: ICDM, pp. 1087–1092 (2017)
20. Shin, K., et al.: Tri-fly: distributed estimation of global and local triangle counts in graph streams. In: PAKDD, pp. 651–663 (2018)
21. Stefani, L.D., et al.: TRIÈST: counting local and global triangles in fully dynamic streams with fixed memory size. TKDD **11**(4), 43:1–43:50 (2017)
22. Tsitsulin, A., Mottin, D., Karras, P., Bronstein, A.M., Müller, E.: NetLSD: hearing the shape of a graph. In: KDD, pp. 2347–2356 (2018)
23. Verma, S., Zhang, Z.: Hunt for the unique, stable, sparse and fast feature learning on graphs. In: NeurIPS, pp. 88–98 (2017)

24. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1), 37–57 (1985)
25. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. CoRR abs/1901.00596 (2019)
26. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)
27. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: KDD, pp. 1365–1374 (2015)