Feature Enhanced Graph Representation Method for Graph Classification

Mohamad Abushofa[1], **Amir Atapour-Abarghouei**[2], **Matthew Forshaw**[1], **A. Stephen McGough**[1]

[1], Newcastle University, School of Computing, UK, Email:{m.e.a.abushofa2,matthew.forshaw,stephen.mcgough}@ncl.ac.uk

[2], Durham University, Department of Computer Science, UK, Email: amir.atapour-abarghouei@durham.ac.uk

[1].

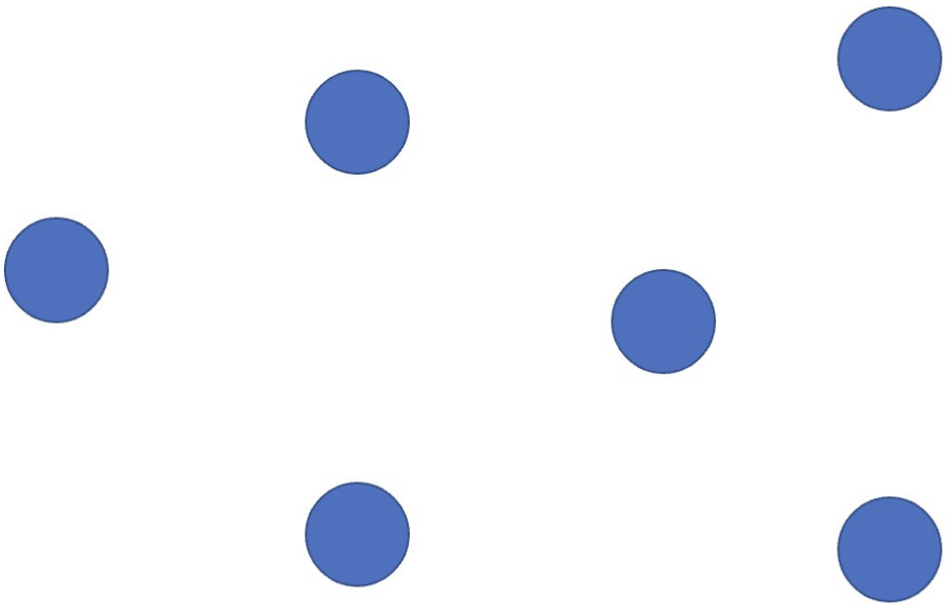
November 7, 2023

[1] 2023 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Introduction

- A graph is formed from a collection of nodes which are connected together via edges. Both nodes and edges may be tagged up with data – called here properties (e.g., in a social network, nodes representing people may have properties of name and age).
- Graph presentations are divided into two categories: Graph Neural Networks (GNNs)Parametric approaches and graph descriptors(non-parametric methods)
- We work on Non-parametric methods to do the Graph classification

Introduction



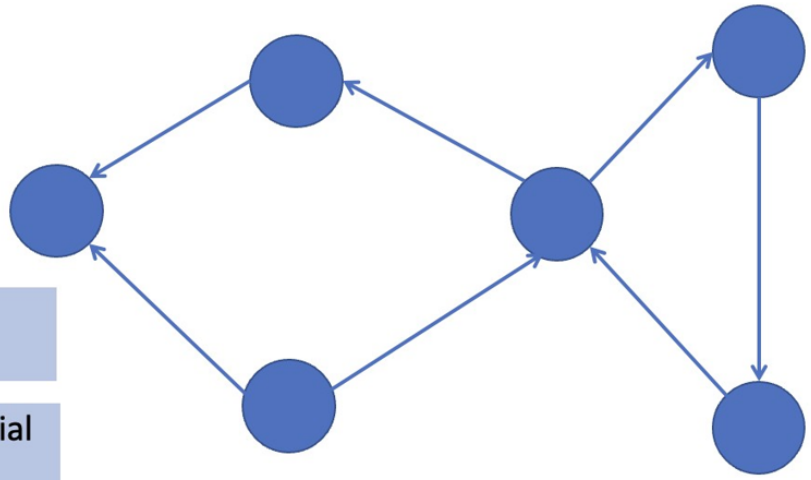
Atom in a molecule

User in a social network

Item in ecommerce network

Protein in PPI

Introduction-Directed Graphs



Bond in a molecule

Friendship in a social network

Transaction in ECN

⋮

Path in traffic network

Atom in a molecule

User in a social network

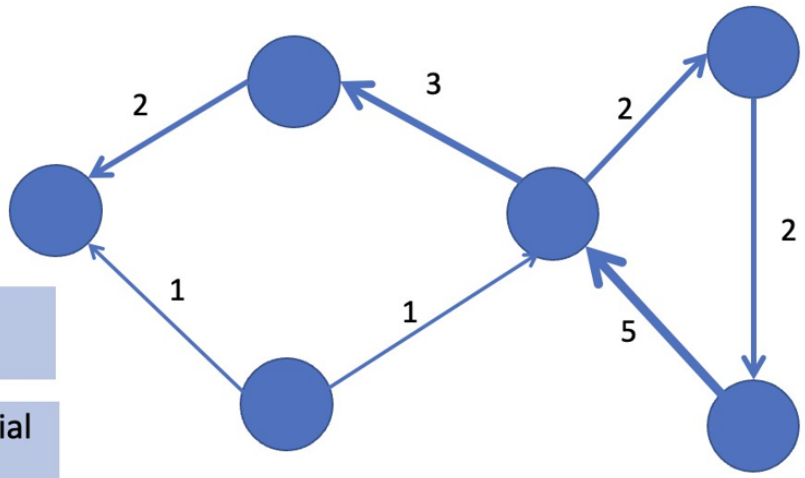
Item in ecommerce network

⋮

Protein in PPI

$G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ and $E \subseteq V \times V$

Introduction-Directed Weighted Graph



Bond in a molecule

Friendship in a social network

Transaction in ECN

Path in traffic network

Atom in a molecule

User in a social network

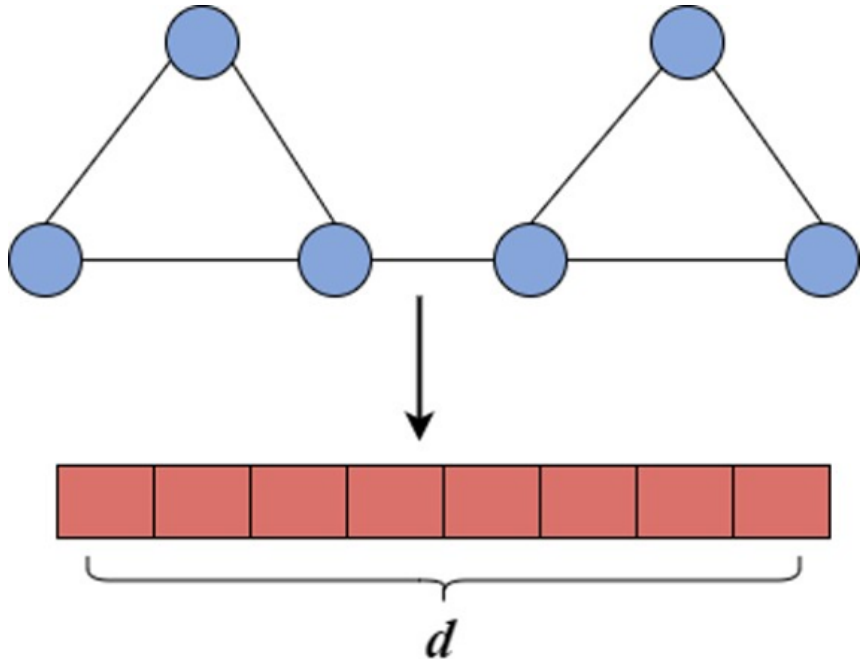
Item in ecommerce network

Protein in PPI

$G = (V, E, W)$ where $V = \{v_1, v_2, \ldots, v_n\}$ and $E \subseteq V \times V$

Introduction-Graph Representation Problem

- Given a graph representing real-world entities and their relationships
- The goal is to design a function $f : G \rightarrow \mathbb{R}^d$
- To perform any downstream Machine Learning task



Our main contributions

- We enrich the graph embedding by incorporating aggregated original feature information associated with each node with local and global graph measures.
- We evaluate the proposed method in different experimental settings, including graph classification, sensitivity, and the evolution of nodes and edges in a graph.
- We have made the source code publicly available to enable better reproducibility of the results

Motivation and Requirements

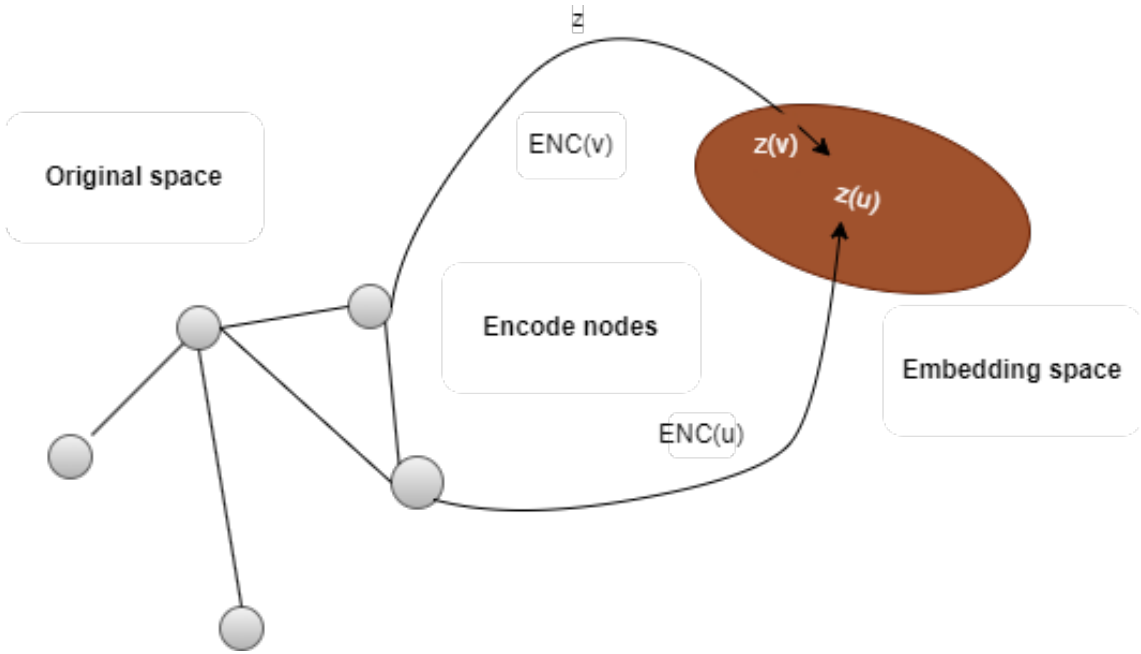
- Graph Machine learning applications have found their way into the healthcare sector, so there are necessary to find robust representation
- There are two important factors that should be addressed, such as accuracy, scalability.
- The computational complexity of graph representation approaches often grows exponentially with the number of nodes, existing approaches are often unsuitable for massive graphs.

Requirements

Our approach is driven by the following requirements:

- Scalability: the new approach should be highly scalable, to graphs of millions of vertices / edges, and capable of doing the classification in a reasonable time. Ideally, the approach should be portable to a many-core or distributed graph processing system to help with scalability.
- Sensitivity To Graph Size: our approaches should take the size and order of the graphs into consideration.
- Sensitivity To Topologies: it should be able to detect the difference between graphs which are highly structurally and topologically similar.
- Accuracy: the approach should provide a graph embedding that results in a high level of accuracy.

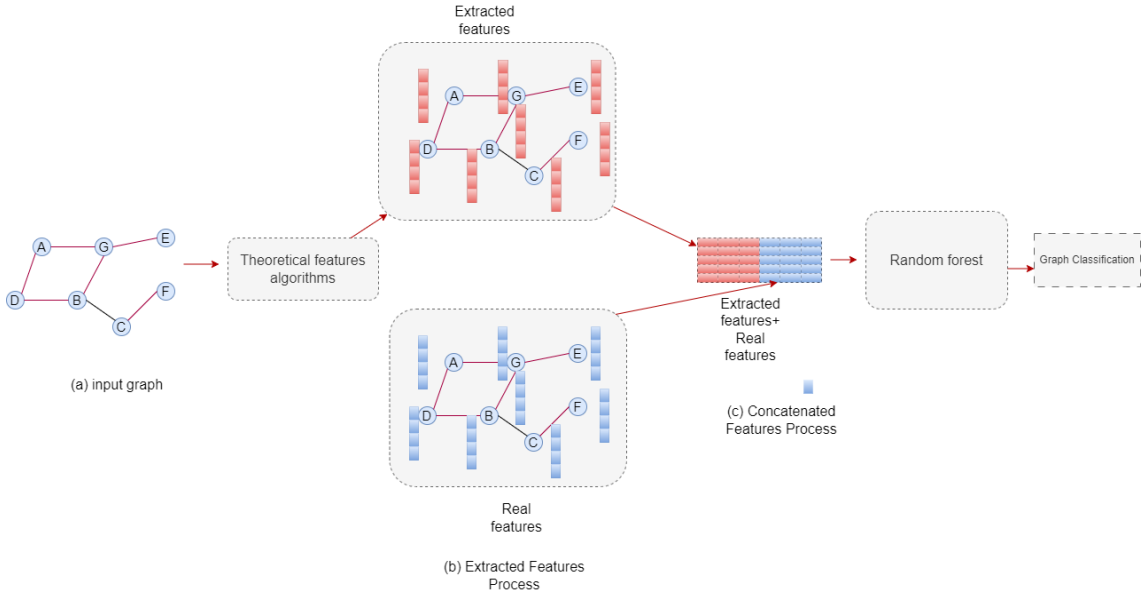
Graph Representation Framework



Non-parametric approaches

After obtaining the graph embedding, we utilize a non-parametric classifier

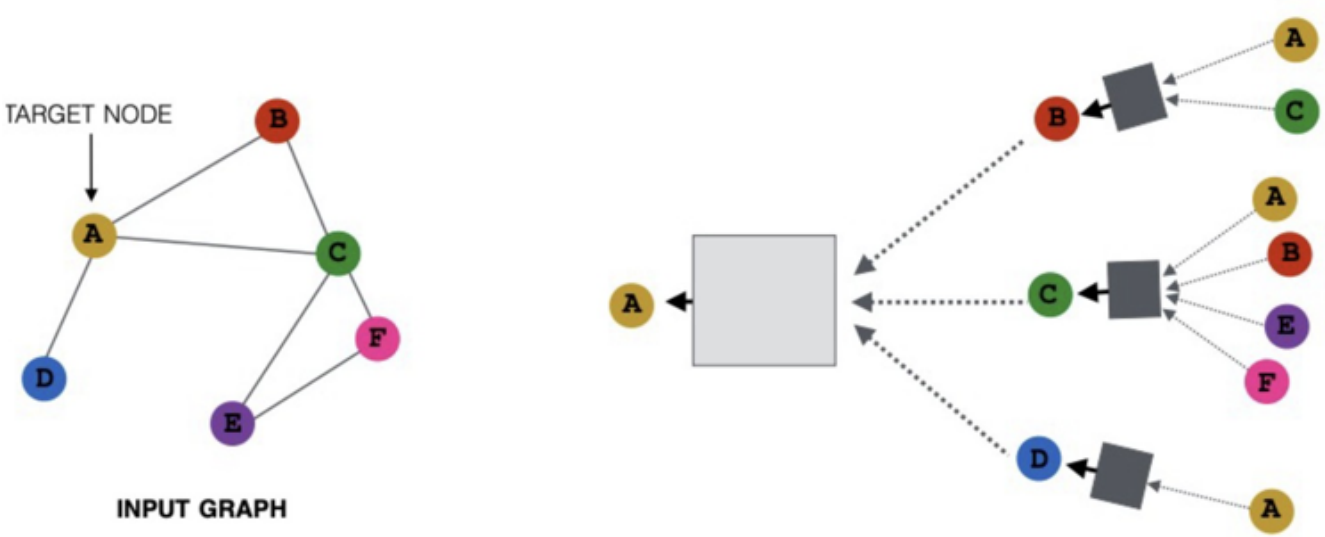
- K-Nearest
- Decision Trees
- Random Forest:
- Support Vector Machines (SVM):
- In our methodology, we employ the random forest algorithm to achieve a more robust data representation.



Parametric approaches

In graph representation, to encode the the graph, we can use different parametric approaches (GNNs baseline) such as

- Graph neural networks(GNNs)
- Graph convolutional networks (GCNs)
- GraphSAGE



Hamilton et al., 2017

$$m_v^{l-1} = f^l(h^{(l-1)}(u)) \quad \text{for } u \in N(v) \tag{1}$$

$$h_v^l = g^l(h^{(l-1)}(v), m_v^{l-1}) \tag{2}$$

Methodology

- Generating graph topological
- Generating graph fingerprint
- Aggregated node properties

Generating graph topological descriptors

- Generating graph topological descriptors
  - Local features:
    - Node degree
    - Clustering coefficient
    - the average degree of neighbours
    - Average clustering of neighbourhood
  - Global features:
    - The largest five eigenvalues
    - Total number of nodes
    - Total number of edges
    - Number of connected components

To encode the local and global properties of the graph, we follow these steps:

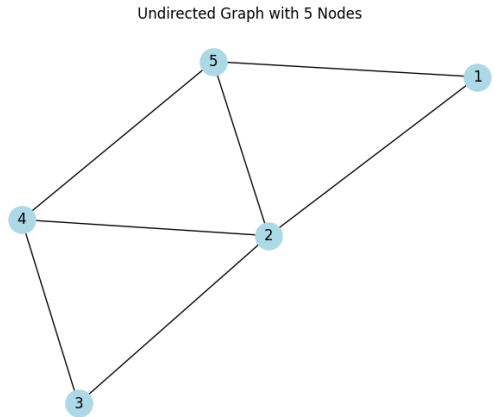
- We initially compute the local features for each vertex.
- Statistics are then computed for each of the four local features. We use the statistics of median, mean, standard deviation, skewness, and kurtosis to form the graph local descriptor.
- These can be concatenated into a vector of twenty values.

Local features

The FEGR approach extracts four features from each vertex within a graph.

- Through experimentation, we have determined that the following four feature metrics give the best balance between topological sensitivity and run-time.
- Other metrics could also be used if other characteristics of a graph are important.
- For each of the four vertex features listed below, a value is extracted for each vertex $v \in V$
- **Node Degree (δ):** The number of connections a given node has to other nodes in the network.

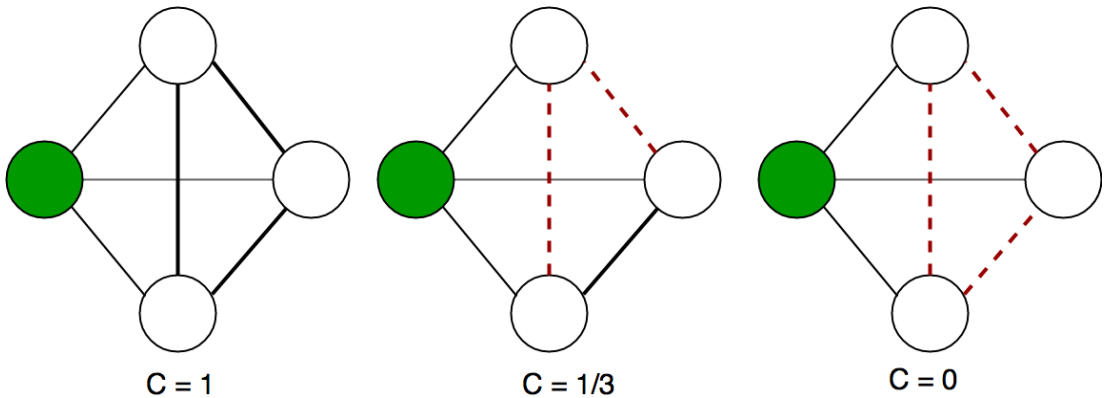
- Node 1 has a degree of 2
- Node 2 has a degree of 4
- Node 3 has a degree of 2
- Node 4 has a degree of 3
- Node 5 has a degree of 3



Local features

- **Clustering coefficient** c**:** the clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together.
- The clustering coefficient c is calculated as follows:

$$c_u = \frac{|(v_1, v_2) \in E : v_1, v_2 \in N|}{2^{d_u}}$$



Local features

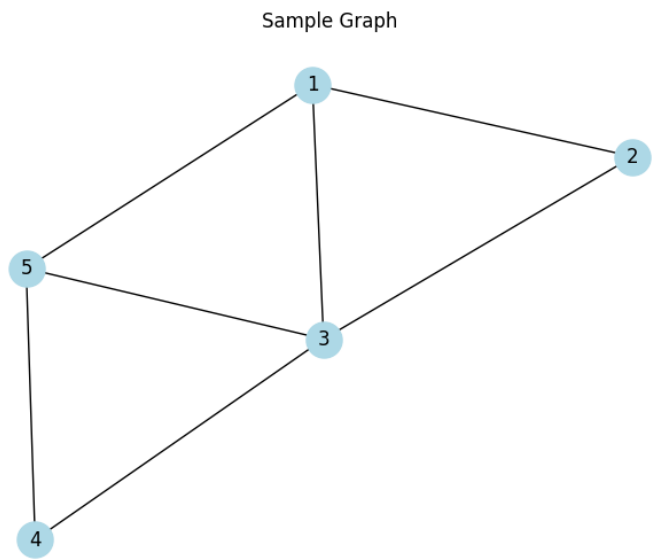
- **The average degree of neighbors**, it denoted as k, is the average degree of the neighborhood of each node. It can be calculated as:

$$k_{nn,i} = \frac{1}{|N(i)|} \sum_{j \in N(i)} k_j$$

  where $N(i)$ represents the neighbors of node i and k_j is the degree of node j belonging to $N(i)$.
- **Average clustering of the neighborhood:** The average clustering score of the neighborhood is determined for each vertex by computing the mean of all the local clustering scores within the vertex's neighborhood.

Global Features

- **The largest five eigenvalues** $\{e_1, \ldots, e_5\}$**.** We keep the top five values.



Algorithm 1: Calculate Largest Five Eigenvalues

Data: Graph G

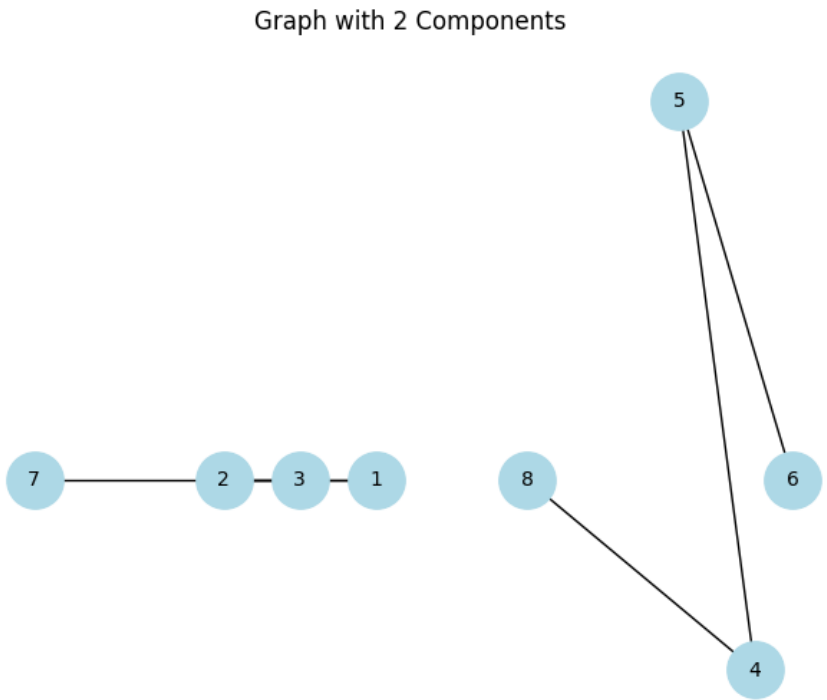
Result: Largest Five Eigenvalues

1 eigenvalues ← np.linalg.eigvals(nx.adjacency$_m$atrix(G).*toarray*()) *sorted$_e$igenvalues*← sorted(eigenvalues, reverse=True);

2 largest$_f$*ive$_e$igenvalues*← sorted$_e$*igenvalues*[: 5]

Global Features

- **Total number of nodes,** $N = |V|$.
- **Total number of edges,** $L = |E|$.
- **Number of connected components,** C. This is the total number of components within the graph, with a component being a subgraph in which there is a possible path between every vertex, while vertices in different components have no possible path between them.



Generating Graph Fingerprint

Statistical terms

$$\{\text{med}(\delta), \bar{\delta}, \text{sd}(\delta), \text{skew}(\delta), \text{kur}(\delta),$$
$$\ldots,$$
$$\text{med}(n), \bar{n}, \text{sd}(n), \text{skew}(n), \text{kur}(n)\}$$

where $med(\square), \bar{\square}, sd(\square), skew(\square)$ and $kur(\square)$ are the median, mean, standard deviation skewness and kurtosis of $\square$. We can then concatenate the four global features to this fingerprint:

Local and global features combination

$$\{med(\delta), \bar{\delta}, sd(delta), skew(\delta), kur(\delta),$$
$$\ldots,$$
$$med(n), \bar{n}, sd(n), skew(n), kur(n),$$
$$e_1, e_2, e_3, e_4, e_5, N, L, C\}.$$

Aggregated Node Properties

- Mapping node i to a vector of values $[x_{i,1}, x_{i,2}, ..., x_{i,n}]$. gives us a vector of length n for each of the m nodes in the graph.
- aggregate this data down to a simple vector. We could aggregate this down using the same approach as used for the local node properties (median, mean, standard deviation, skew and kurtosis for each of the n elements)
- sum the vectors have given promising results. Therefore, we define the aggregate properties as

Aggregating properties

$$[\sum_{i=1}^{m} x_{i,1}, \sum_{i=1}^{m} x_{i,2}, ..., \sum_{i=1}^{m} x_{i,n}].$$

Algorithm 2: Local Feature Extraction Algorithm

Input: $G_1, G_2, \ldots, G_k$ (a set of graphs)
Output: $\{FG_1, FG_2, \ldots, FG_k\}$ (a set of node $\times$ feature matrices)

for $j \in \{G_1, G_2, \ldots, G_k\}$ **do**
    $FG_j = []$ // Initialize feature matrix for G_j // Extract features for all nodes in G_j
    for $i \in V_j$ **do**
        $FG_j = FG_j \cup \{\delta, c_i, \mathrm{k}, \mathrm{c}_a, \}$

// Return a set of node $\times$ feature matrices **return** $\{FG_1, FG_2, \ldots, FG_k\}$

Algorithm 3: Feature Aggregation Algorithm

Input: $\{FG_1, FG_2, \ldots, FG_k\}$ (a set of node $\times$ feature matrices)
Output: $\{s\tilde{G}_1, s\tilde{G}_2, \ldots, s\tilde{G}_k\}$ (a set of "signature" vectors for the graphs)

for $j \in \{FG_1, FG_2, \ldots, FG_k\}$ **do**
    $s\tilde{G}_j = []$ // Initialize "signature" vector for G_j
    //For each feature column in FG_j, compute a set of aggregates
    for $feat \in FG_j$ **do**
        $s\tilde{G}_j =$
        $s\tilde{G}_j \cup \{\mathrm{median}(\mathrm{feat}), \mathrm{mean}(\mathrm{feat}), \mathrm{stdev}(\mathrm{feat}), \mathrm{skewness}(\mathrm{feat}), \mathrm{kurtosis}(\mathrm{feat})\}$

// Return a set of "signature" vectors for the graphs **return** $\{s\tilde{G}_1, s\tilde{G}_2, \ldots, s\tilde{G}_k\}$

Algorithm 4: Global Feature Extraction Algorithm

Input: $G_1, G_2, \ldots, G_k$ (a set of graphs)
Output: $\{F_g G_1, F_g G_2, \ldots, F_g G_k\}$ (a set of node $\times$ feature matrices)

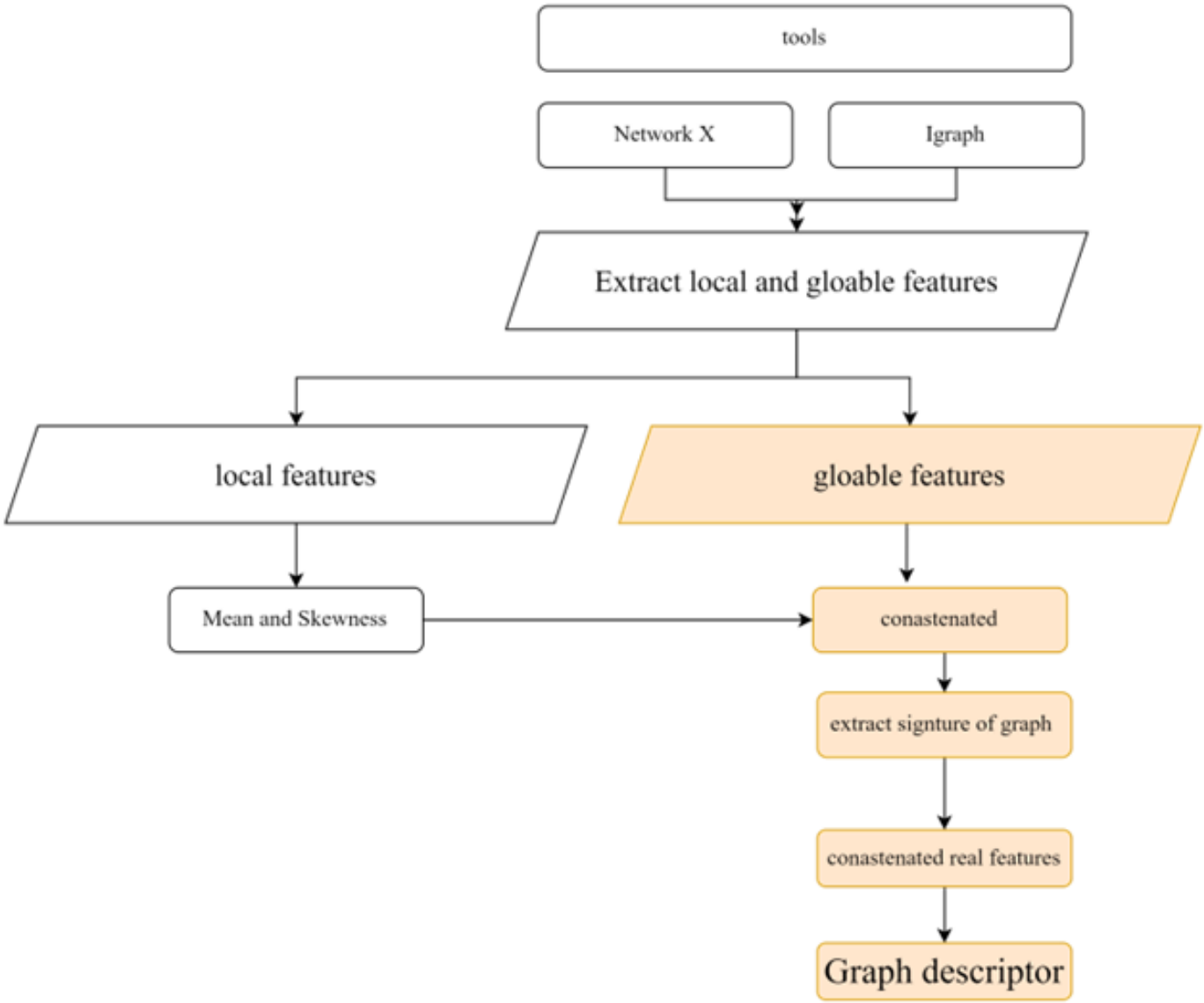
for $j \in \{G_1, G_2, \ldots, G_k\}$ **do**
    // Extract global features for whole graph G_j
    Compute the largest five eigenvalues $\{e_1, e_2, e_3, e_4, e_5\}$ of G.
    $N = |V|$
    $L = |E|$
    $C =$ Number of connected components in G
    // Return the computed features **return** $\{e_1, e_2, e_3, e_4, e_5\}, N, L, C$
// Return a set of global features for each graph **return** $\{FG_1, FG_2, \ldots, FG_k\}$

Algorithm 5: Graph Processing and Classification

Input: List of graphs $\{G_1, G_2, G_3, \ldots, G_n\}$

Initialize empty lists, *labels*, *graphs*,*features*=[],[],[]
for *each j in* $\{G_1, G_2, G_3, \ldots, G_n\}$ **do**
    Append *d.y.item*() to *labels* Convert *d* to a networkx graph with specified attributes
    Append the graph to *graphs* list
    Calculate features by summing node attributes *d.x* along axis 0
    Append features to *features* list
Initialize an empty list *graph_signatures*
for *each index, g in enumerate(graph_list)* **do**
    Compute signature *sig* for *g*
    //where sig are list of local and global features
    Get the corresponding networkx graph *nx_g*
    Calculate adjacency matrix *adj* for *nx_g*
    Extend *sig* with features from *features*[*index*]
    Append *sig* to *graph_signatures* list
// Apply a random forest classifier to the graph signatures *acc* = apply_RF_Grid(*graph_signatures*, *labels*)

Methodology diagram



Baseline

- FGSD: A spectral graph descriptor that uses spectral distances among pairs of nodes.
- NetLSD: Uses the idea of heat kernels.
- HOSD: Uses subgraphs' counting with compression.
- NetSIMILE: Uses statistical node and graph-level properties.
- Shortest Path Kernel: Based on pair-wise distances.
- WL kernel: Based on the idea of color refinement based on nodes' neighborhood.

Datasets for Graph Classification

- **MUTAG:** It contains a total of 188 graphs with an average of 17.93 nodes and 19.79 edges per graph.
- **Proteins:** It includes 1113 graphs with an average of 39.06 nodes and 14.69 edges per graph.
- **PTC:** It comprises 344 graphs with an average of 25.56 nodes and 72.81 edges per graph.
- **AIDS:** It contains 2000 graphs with an average of 15.58 nodes and 16.19 edges per graph.
- **NCI1:** It consists of 4110 graphs with an average of 29.87 nodes and 32.3 edges per graph.
- **NCI109:** It comprises 4110 graphs with an average of 29.56 nodes and 32.13 edges per graph.
- **D & D:** It includes 1178 graphs with an average of 284.3 nodes and 715.65 edges per graph.

The number of classes in these datasets is two. These are benchmark graph classification datasets available on the Torch Geometric website.

Experimental setup and Results

- Results:We can observe from the results (Table I) that the proposed method outperforms all other methods on PROTEINS and DD datasets while the results on the AIDS dataset are identical to that of DGSD. We outperform all other methods on AIDS as well. On the remaining dataset, we achieved results within 1% of the top result on MUTAG, and NCI1 and within 3% on PTC and NCI109 datasets. These results clearly show that graph global features positively contribute to the graph representation.

Classification Accuracy Comparison

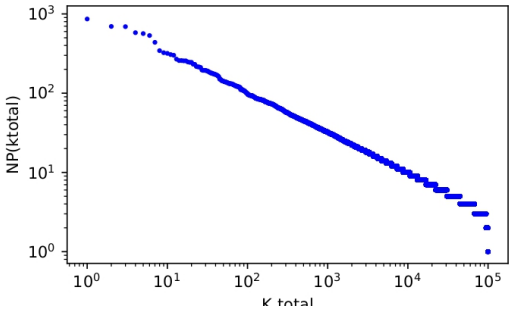
Dataset	SP	NHK	EHK	GK	NetSimile	FGSD	NetLSD		DGSD	**Ours**
							$w(g)$	$h(g)$		
Mutag	86.60	85.06	85.37	77.01	83.42	**88.26**	82.40	83.31	87.70	87.80
PTC	59.00	60.58	57.54	57.56	55.80	60.70	57.22	53.49	**61.32**	58.20
Proteins	74.12	74.29	59.56	73.22	69.71	70.25	68.10	72.14	73.68	**76.10**
NCI1	71.65	75.52	50.04	58.12	68.87	**79.75**	61.94	67.25	73.48	78.9
NCI109	71.48	75.23	50.37	58.97	67.45	**80.44**	60.38	64.64	72.01	77.67
AIDS	99.24	99.2	99.60	98.75	97.95	98.5	93.7	99.69	**99.8**	**99.8**
D&D	77.94	75.81	58.65	>D	73.86	75.9	70.21	72.33	78.52	**78.54**

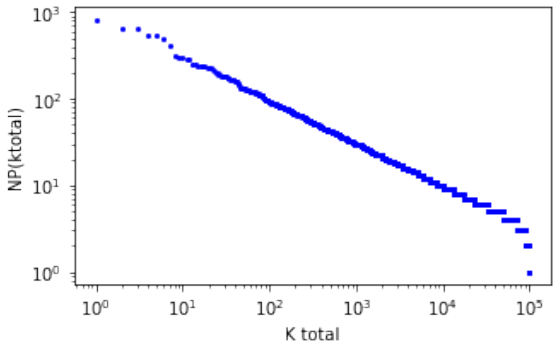
Table: Classification accuracy comparison. >D indicates experimental time exceeds a day. We report the state-of-the-art results from DGSD because we set our experimental procedure similar to it to provide a fair comparison.

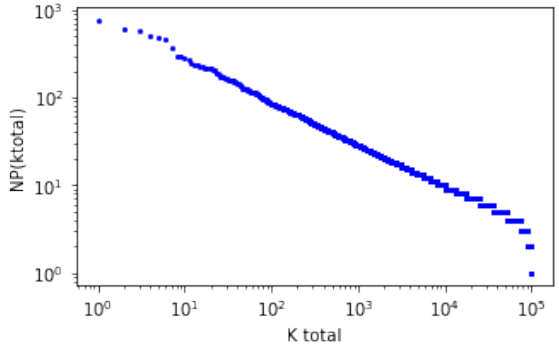
Method Sensitivity Evaluation

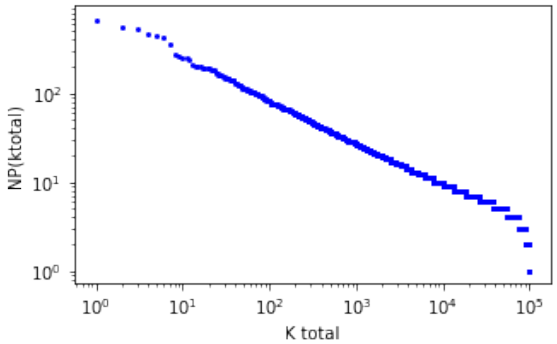
- we generated a graph with Barabasi Abert model with 100,000 nodes and 300,000 edges.
- We consider six network rewiring experiments, which are rewiring 20,000, 40,000, 50,000, 60,000, 80,000, and 100,000 edges in the previously generated original graph. Also,
- We can clearly see that as the number of edges in the rewiring increases, the distance between the embeddings increases. This asserts the proposed descriptor is very sensitive to the graph topology, which means it can discriminate between similar graphs very well.

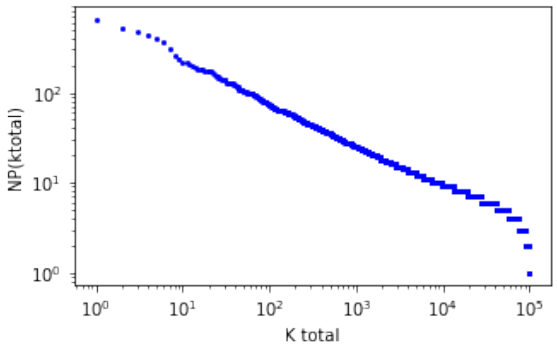
Sensitivity to Rewiring Edges



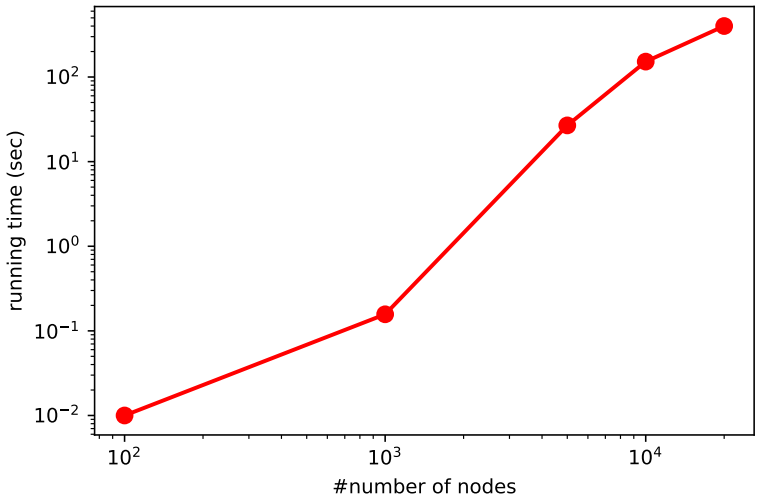


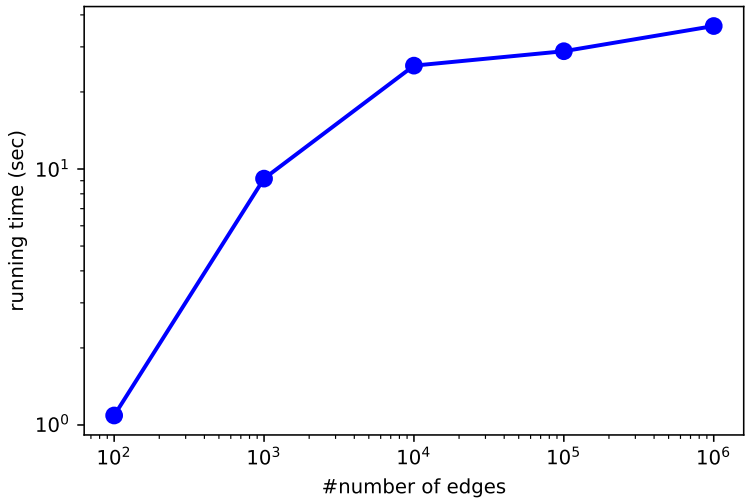


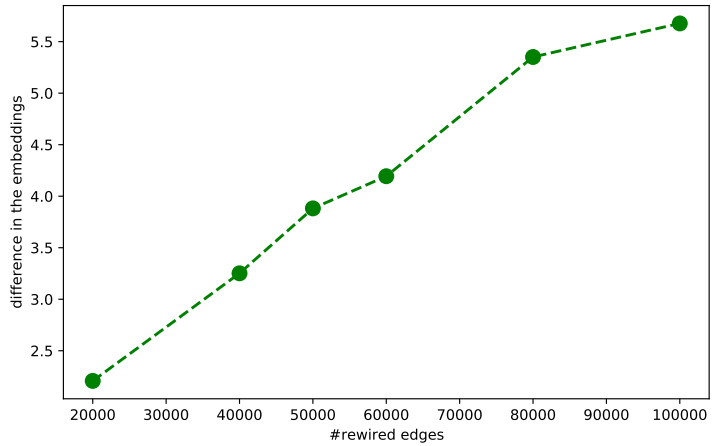




Run time Analysis







Conclusion

- The paper introduces innovative graph representation methods designed for graph classification tasks.
- It extensively investigates a range of graph theory concepts and frameworks, culminating in the identification of nine measures for constructing a comprehensive graph descriptor.
- These measures are adept at capturing both local and global information within the graph structure.
- In addition to the theoretical features, the method utilizes aggregated node-level attributes to create a more expressive representation.
- Comprehensive evaluations conducted on multiple graph classification datasets demonstrate the effectiveness and practicality of the proposed approach.