This is a peer-reviewed, accepted author manuscript of the following conference paper: Kyosev, I, Paun, I, Moshfeghi, Y & Ntarmos, N 2021, Measuring distances among graphs en route to graph clustering. in X Wu, C Jermaine, L Xiong, et al. (eds), 2020 IEEE International Conference on Big Data. IEEE, Piscataway, NJ, pp. 3632-3641. https://doi.org/10.1109/BigData50022.2020.9378333

Measuring Distances Among Graphs En Route To Graph Clustering

Ivan Kyosev*, Iulia Paun*, Yashar Moshfeghi†, Nikos Ntarmos*

*School of Computing Science, University of Glasgow, UK
Email: 2092065k@student.gla.ac.uk, {iulia.paun, nikos.ntarmos}@glasgow.ac.uk
†Department of Computer and Information Sciences, University of Strathclyde, UK
Email: yashar.moshfeghi@strath.ac.uk

***Abstract*—The graph data structure offers a highly expressive way of representing many real-world constructs such as social networks, chemical compounds, the world wide web, street maps, etc. In essence, any collection of entities and the relationships between them can be modelled using a graph, thus preserving more information about the real-world objects than a simple vector space model. An issue that arises when operating on collections of graphs, however, is that most statistical analysis and machine learning methods expect their input data to be in the form of multidimensional vectors, where all items can be compared with each other using well-understood metrics such as Euclidean or Manhattan distance. This paper presents a variety of approaches for computing distances between graphs with known node correspondence, with the aim of applying those measures alongside clustering algorithms to discover patterns in a given dataset. The performance of each distance measure is then evaluated through its ability to identify communities of graphs with similar features. We show that because the considered distance metrics highlight different structural properties, the method that produces the highest quality result will depend on the characteristics of the processed graph population.**

***Index Terms*—Graph Distance, Graph Mining, Graph Clustering.**

I. Introduction

In recent years, machine learning algorithms have become essential components for the operation of a variety of systems that users interact with on a daily basis. Such applications continuously gather data from customer interactions, sensors, network traffic, system logs, etc., to construct internal representations for the relevant objects of interest. These representations commonly take on the form of multidimensional feature vectors, where the values found at each index are either discrete – expressing some form of label or class associated with the target objects – or continuous – denoting the intensity of a particular feature [1]. Such models enable the embedding of objects into $\mathbb{R}^n$ space as simple points, which can then be processed using traditional statistical analysis techniques.

One of the main disadvantages of the vector space model is that the values expressing the points' positions in the individual dimensions are assumed to be independent [2] – i.e., there is no relationship among the expressed features. This may not be the case for the original objects, however. In that case, the N-dimensional vector representation is insufficient to portray the characteristics observed within the target objects accurately, e.g. information about the direct and indirect correlations amongst an object's features would be lost.

An alternative way of modelling real-world objects, along with the relationships among their components, can be achieved using graphs – a set of *vertices*, plus *edges* joining vertex pairs. However, the main problem that emerges from operating on a collection of graphs, is that common statistical analysis methods can no longer be applied directly [3]. An essential requirement for a large quantity of machine learning and data mining methods is the ability to calculate a measure of distance between the individual elements of an analysed population. When the data being processed is in the form of feature vectors, i.e. representing points in $\mathbb{R}^n$ space, then well-known distance metrics such as Euclidean or Manhattan can be used. If the data is not in the form of feature vectors but rather in graph form, it is necessary to define a measurement of the dissimilarity between the data items that takes their unique properties into account.

Moreover, datasets consisting of graphs with known node correspondence arise quite naturally in several use cases; databases of graph-structured (user/item) profiles [4], collections of instances from a common ontology [5], flows through network graphs [6], etc., are all examples of such datasets. We have chosen to focus our research on this sub-space of the general graph distance/clustering space. There are two reasons for our decision. First, this is an area that hasn't received much attention in the relevant literature as it is usually deemed "easy" (a statement refuted by this work). Second, it addresses a pressing and fairly open problem in an era where graph databases/processing proliferate.

This paper aims to survey existing methods for computing distances between graphs with known node correspondence (i.e. all graphs have an equal number of vertices with known IDs, while only the connectivity changes), to propose new methods based on recent graph mining techniques, and to make a first step towards identifying the best fitted family of methods for various types of graphs. The identified methods are evaluated on their ability to discover logical communities of graphs with similar features. The methods vary greatly in

This work is supported by the Erasmus+ Programme of the European Union under the PRIMES project (no. 2016-1-UK01-KA201-024631) and by the UK Government through an EPSRC grant (no. 509668).

terms of their approach and also highlight different structural properties – making the preferred choice dependent on the specific characteristics of the examined graphs.

The contributions made by this paper are:

C1: A review of existing methods for measuring the distance between graphs with known node correspondence (§ III).

C2: Novel algorithms for computing distances between graphs with known node correspondence, based on recent graph mining techniques (§ IV).

C3: A method for comparing the quality of cluster assignments produced using different distance functions (§ V-C).

C4: An experimental evaluation of the considered graph distance metrics, both in terms of the quality of the produced clustering over different datasets and their performance when operating on larger input datasets (§ V-D).

II. Graph Distance and Clustering

It is first necessary to provide a formal definition for a graph as well as to outline the essential requirements for any graph distance measure. A graph G may be written as $G = (V, E)$, where V is the vertex set and E is the edge set, such that $E \subseteq \{(u, v) : u, v \in V\}$, i.e., each edge $e \in E$ links two vertices from V. An edge $e = (u, v)$ may be directed, in which case u denotes the *source* vertex and v the *destination* vertex. Furthermore, an edge can be weighted or unweighted – implicitly assuming unit weight. The graph representation that we employ in this paper is the adjacency matrix A – an $n \times n$ matrix, where $n = |V|$ (the size of the vertex set). The individual elements of A at position A_{ij} are either 0, if there is no edge from vertex i to vertex j, or equal to the weight of the edge from i to j. We also consider the diagonal matrix D – an $n \times n$ matrix of zeros, except for the values along the diagonal D_{ii}, denoting the number of edges for which i is a source vertex – i.e., the out degree of vertex i.

When one requires a measure of the dissimilarity between a pair of graphs, it is necessary to define a function $d(G_1, G_2)$, which provides a meaningful indication of the graphs' structural differences. The following characteristics must hold true for any such function d:

- Symmetry: $d(G_1, G_2) = d(G_2, G_1)$
- Identity: $d(G_1, G_1) = 0$
- Non-negativity: $d(G_1, G_2) \geq 0$

Additionally, the distance measure would need to be efficient to compute and scalable when being used as part of a machine learning procedure, operating on a large data set.

A key constraint on the scope of this paper is that we will only consider distance measures between graphs with known node correspondence. In this scenario, all graphs within a particular data set $\mathcal{G} = \{G_1, G_2, ..., G_g\}$ have an equal number of nodes with known IDs - i.e., the vertex set V of all graphs in $\mathcal{G}$ is identical. The result returned by $d(G_1, G_2)$ can then more precisely be interpreted as the difference in the connectivity of the constant vertex set V found in both G_1 and G_2. This model aims to emulate the set of graph representations for a collection of objects with common features and varying relationships between those features. While there are algorithms for measuring the similarity of graphs with general structures (typically based on subgraph isomorphism [7] or finding a maximum common subgraph [8]), the computational cost of these methods makes them impractical for large data mining tasks [9].

Four different approaches for measuring the distance between graphs with known node correspondence were identified in the literature:

1) Methods based on **edit distance** (§ III-A), where one graph is transformed into another through a series of operations (insertion, deletion, substitution), each associated with a cost. The aim is to find the sequence of operations that minimise the cost of matching the graphs.
2) **Node affinity** methods (§ III-B and § IV-A), where two nodes are considered to be similar if their neighbourhoods are similar – the distances between the graphs can be calculated by aggregating the difference between the corresponding pairs of nodes.
3) Methods based on **feature extraction** (§ IV-B and § IV-C), where the corresponding pairs of nodes in different graphs are compared on properties such as: node in-/out-degree, number of edges in the nodes' neighbourhood, etc.
4) Methods that rely on **deep learning** (§ IV-D) to produce an embedding of a graph's nodes into $\mathbb{R}^n$ space.

Representatives from each of those categories were selected and applied alongside clustering algorithms to determine their ability to differentiate between communities of graphs with similar characteristics.

III. Existing Distance Measures

This section discusses algorithms that directly address the problem of computing a measure of distance between graphs with known node correspondence.

A. Graph Edit Distance

The collective cost of performing a series of graph edit operations that aim to transform one graph into another is known as the Graph Edit Distance (GED) [10]. This method of measuring the level of distortion between a pair of objects, using a set of edit operations, was first proposed for string representations [11] and was later extended to more general data structures such as trees [12] and graphs.

Given two graphs, one of them being the source $G_1(V_1, E_1)$ and the other is the target $G_2(V_2, E_2)$, GED aims to transform G_1 into G_2 through a number of modifications. These modifications typically consists of insertions, deletions and substitutions of both vertices and edges. Then, an edit path $\lambda(G_1, G_2) = \{e_1, e_2, ..., e_k\}$ is a set of k edit operations that completely transform G_1 into G_2. For every such pair of graphs there is an infinitely large set of possible edit paths $\gamma(G_1, G_2)$. To identify the best one, it is necessary to introduce a cost $c(e)$ for each edit operation, measuring the impact of performing the transformation. Low and high costs would

Algorithm 1 Graph Edit Distance

INPUT: edge files of $G_1(V, E_1)$ and $G_2(V, E_2)$
A_1 = getAdjacencyMatrix(G_1)
A_2 = getAdjacencyMatrix(G_2)
$S = A_1 - A_2$
$dist(G_1, G_2) = \sum_i \sum_j |S_{ij}|$
return $dist$

correspond to small and large modifications of the source graph, respectively. The cost of an edit path is then calculated as the sum of the cost of the edit operations it consists of, and so the edit distance between G_1 and G_2 is equal to the lowest cost edit path $\lambda(G_1, G_2)$ in $\gamma(G_1, G_2)$:

$$GED(G_1, G_2) = \min_{\lambda \in \gamma(G_1, G_2)} \sum_{e_i \in \lambda} c(e_i)$$

The cost functions $c(e)$ for each of the possible edit operations need to follow a set of conditions to ensure that only a finite set of edit paths need to be explored to compute the GED. These include: non-negativity, where $c(e) \geq 0$ for all edge and vertex edit operations (only substitution/renaming operations may have a cost of 0); triangle inequality, i.e., $c(a \rightarrow c) \leq c(a \rightarrow b) + c(b \rightarrow c)$; and symmetry – $c(e) = c(e^{-1})$, where e^{-1} is the inverse operation of e. When considering unlabeled graphs, it is sufficient to attribute unit cost to any insertion or deletion of nodes or edges. If this is not the case, the cost of these modifications is defined with respect to the label alphabet.

In the scenario where GED is being used to compute the distance between a pair of graphs with an equal number of vertices and a known correspondence between them, it is only necessary to calculate the cost of equating the two edge sets. The optimal edit path from G_1 to G_2 would therefore consist of the deletion of all edges in G_1, not present in G_2, and the insertion of all edges in G_2, not present in G_1. This can be modelled by subtracting the adjacency matrix of G_1 from that of G_2 and taking the sum of the absolute values of all entries in the resulting matrix (alg. 1).

B. DeltaCon

In the field of graph mining, there is a plethora of algorithms which focus on calculating node affinities – i.e., one or more metrics per node computed using the connectivity of the analysed graph. Examples include the popular PageRank algorithm [13], Hyperlink-Induced Topic Search (hubs and authorities) [14], SimRank [15], and others. Their intended purpose is to compare the individual nodes within a graph and to discover key structural points. This general approach of computing a series of values per node can also be adapted to calculating the distance between graphs with known node correspondence by measuring the differences between the relevant pairs of nodes.

DeltaCon (DC) [16] presents an example application of this technique. As its name suggests, it is a measure of the change in connectivity between two graphs with an identical vertex set. DeltaCon functions by first processing each graph individually and computing the pair-wise affinities for its nodes using Fast Belief Propagation (FaBP) [17] – a technique similar to personalised Random Walks with Restarts (RWR) [18]. The output of FaBP is an $n \times n$ affinity matrix ($n = |V|$), where each entry s_{ij} of the matrix indicates how much influence node i has on node j - this value will be large if there are many, short, heavily weighted paths from i to j. FaBP's main advantages are its low computation cost and that it provides intuitive results for each node, taking into account not only direct neighbours but also 2-, 3-, and k-step away neighbours, with decreasing weights. The following formula is used to derive the $n \times n$ affinity matrix of a graph G using FaBP:

$$S = [I + \epsilon^2 D - \epsilon A]^{-1}$$

where I is the identity matrix, A and D are the adjacency and diagonal matrices of G, an ϵ is a positive constant, such that $\epsilon = 1/(1 + max_i(d_{ii}))$.

Once the affinity matrices S_1 and S_2 have been computed for a pair of graphs G_1 and G_2, DeltaCon uses the Root Euclidean distance (RootED) to produce a single scalar value, indicating the dissimilarity of the two graphs:

$$d = RootED(S_1, S_2) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{1,ij}} - \sqrt{s_{2,ij}})^2}$$

Koutra et al. [16] chose this measure over regular Euclidean distance, as RootED boosts the node affinities (their values range from 0 to 1), allowing the algorithm to detect even small changes in the graphs – using ED results in low distance values even if the graphs differ substantially.

Because DeltaCon is presented as a means of computing the similarity of two graphs, the algorithm's last stage involves converting the calculated distance value d into a score, ranging from 0 to 1, using the formula: $sim = \frac{1}{1+d}$. A similarity score of 1 would, therefore, indicate that the graphs are identical, while a score of 0 would occur when the first graph is a clique, the second graph has an empty edge set, and the size of the common vertex set approaches infinity. For this paper, when referring to DeltaCon as a measure of distance, this final step is omitted.

IV. Graph Mining Algorithms

This section presents a series of graph mining algorithms, which analyse the structural properties of a graph and produce an embedding of its nodes into $\mathbb{R}^n$ space.

A. SimRank

SimRank (SR) [15] is another node affinity-based method, which aims to compute the similarity between all possible pairs of nodes within a graph. It is based on the principle that "two objects are similar if they are referenced by similar objects". Unlike Fast Belief Propagation [17], which assigns high affinity scores for a pair of nodes a and b if there are multiple, short, heavily weighted paths between a and b, SimRank will return a high similarity value for the pair, if

both nodes have incoming edges from many identical nodes – i.e., creating a path of directed edges from a to b will not increase their similarity.

For any pair of nodes a and b, SimRank denotes their similarity by $s(a, b) \in [0, 1]$. As a base case, the similarity of a node to itself is considered to be one: $s(a, a) = 1$. Otherwise, the following recursive equation is used:

$$s(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$$

where $I(a)$ is the set of in-neighbour nodes for node a, $I_i(a)$ denotes an individual in-neighbour of a, where $1 \leq i \leq |I(a)|$, and C is a constant between 0 and 1. In the scenario where either a or b do not have any in neighbours, their similarity is set to zero: $I(a) = \emptyset \; or \; I(b) = \emptyset \implies s(a, b) = 0$.

The SimRank equation is invoked for each pair of nodes, resulting in a set of n^2 operations for a graph with n nodes. The similarity for any two nodes a and b is therefore equal to the sum of similarities of every possible pair of their in-neighbours, normalised by $|I(a)||I(b)|$. From the above equation it is also clear that SimRank scores are symmetric, i.e., $s(a, b) = s(b, a)$.

In regards to the C parameter, it is treated as a confidence level or a decay factor. In the scenario where nodes a and b both only have a single common in-neighbour c, because the similarity of c with itself is 1, the SimRank equation without the decay factor would also return a value of 1 when comparing a and b. Instead, the calculation becomes $s(a, b) = C \cdot s(c, c)$, indicating that there is less confidence in the similarity of a and b, than there is between c and itself.

Because of the recursive nature of the SimRank equation, in order to compute the scores for the node pairs in a graph G, it is necessary to first assign each of the n^2 pairs an initial similarity value and then iterate until a fixed-point. For each iteration k, the n^2 similarity scores for all node pairs $s_k(\star, \star)$ are computed over the values from the previous iteration $s_{k-1}(\star, \star)$. The initial values are therefore set to be:

$$s_0(a,b) = \begin{cases} 1 \ , \ \text{if } a = b \ , \\ 0 \ , \ \text{if } a \neq b \ . \end{cases}$$

and represent a lower bound on the actual SimRank scores (the values $s_k(\star, \star)$ for each successive iteration are non-decreasing). The similarities at iteration $k + 1$ are then:

$$s_{k+1}(a,b) = \frac{C}{|I(a)|\,|I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s_k(I_i(a), I_j(b))$$

These values are shown to quickly converge [15], with a suggested $k = 5$ iterations and a decay factor of $C = 0.8$.

A more succinct method of expressing the n^2 operations that need to be performed for each iteration can be achieved using the matrix representation for SimRank:

$$S_{k+1} = \max\{C \cdot (W^T \cdot S_k \cdot W), I\}$$

where the entries at S_{ij} denote $s(i, j)$, W is the column normalized adjacency matrix, and I is the identity matrix. In this notation $S_0 = I$.

Algorithm 2 SimRank

INPUT: edge file of $G(V, E)$, number of iterations k, decay factor C
A = getAdjacencyMatrix(G)
W = normalizeColumns(A)
$S = I$
for $i = 1$ to k **do**
    $S = \max\{C \cdot (W^T \cdot S \cdot W), I\}$
return S

B. ReFeX and RolX

Feature extraction is an alternative graph mining technique that is not concerned with propagating node affinities. It is the process of analysing the structural properties of a given graph and computing a set of values per node, indicative of its regional information. This typically comes in the form of node degree, a number of edges entering/leaving a node's neighbourhood, and other characteristics of a node's direct and k-steps away neighbours. Extracting effective features is a crucial step for many machine learning tasks such as outlier detection [19] and node classification [20].

1) Recursive Feature Extraction: The Recursive Feature Extraction (ReFeX) [21] algorithm offers an efficient way of obtaining meaningful features for the nodes of a given graph. It recursively combines local (node-based) and neighbourhood (egonet-based) features to produce regional features that capture " behavioural" information. This provides key insight into the qualities of a node's environment – the *kind* of nodes that connect to it.

ReFeX approaches the problem of feature extraction by computing an $n \times f$ feature matrix F (i.e., calculating f features for each node), using only the structural properties of the graph, without any attribute information about the nodes or edges. The algorithm separates the features it extracts into three categories: **local**, **egonet** and **recursive**, with the first two being used to seed the recursive feature generation. The local feature includes measurements of node degree, while egonet features are computed using the node's ego network. This network is composed of the node itself, its direct neighbours, and all edges in the subgraph consisting of these nodes. The information obtained from the egonet consists of the number of edges it contains, as well as the number of edges entering and leaving the egonet.

2) Role Extraction: RolX [20] is an algorithm that extends recursive feature extraction by making use of the obtained feature matrix to perform role extraction. Role extraction on graphs is the process of assigning role membership to the nodes of a graph, which reflects their structural behaviour; e.g., members of cliques, centres of stars, peripheral nodes, 'bridges" between highly connected subgraphs, etc.

The execution of RolX involves three main phases:

1) In **feature extraction**, the previously described ReFeX algorithm is used, providing RolX with an $n \times f$ feature matrix as a starting point.

2) The Minimum Description Length criterion [22] is then used for **model selection** – i.e., determining the number of roles. Because of this operation, it is not necessary for a user to have prior knowledge about the number of structural roles that are present amongst a graph's nodes.
3) Finally, to perform **feature grouping**, Non-negative Matrix Factorization is applied as a form of soft clustering in the structural feature space. In particular, NMF is used to generate a rank r approximation $RH \approx F$, where F is the $n \times f$ feature matrix produced by ReFeX, H is a $r \times f$ matrix, where the columns specify how membership in a specific role contributes to estimated feature values, and R is an $n \times r$ matrix, where the rows represent a node's membership among the available roles.

C. Simple Feature Extraction

As a means of evaluating the benefits of more advanced node analysis algorithms such as RolX, in the context of measuring graph distance, we also introduce a simple feature extraction method (SFE). This technique involves obtaining only local features from the nodes of an input graph. Specifically, our method computes the in and out degrees of each node and return an $n \times 2$ matrix representation of the graph. The clear advantages of this approach are its low computational cost and low storage requirements - the output matrix grows linearly with the number of graphs nodes, unlike FaBP and SimRank which compute n^2 sized matrices.

D. DeepWalk

DeepWalk (DW) [23] is a data mining algorithm for network structures that aims to learn latent representations of adjacency matrices using deep learning techniques developed for language modelling. The first stage of DeepWalk involves obtaining a text corpus from the input graph. The algorithm treats the set of node IDs as its vocabulary V and generates sentences (sequences of node IDs) through a series of random walks, with each node v_i being used as the starting point for γ walks of length t. Each random walk W_{v_i}, beginning at node v_i, is a stochastic process that traverses a section of the graph by uniformly sampling its next destination from the neighbours of the last visited vertex until a maximum length (t) is reached. Perozzi et al. [23] show that the frequency with which vertices appearance within short random walks, resembles that of word frequency in natural languages – enabling modelling techniques in that domain to be re-purposed for graph analysis.

In its second stage, DeepWalk uses the gathered corpus along with an initial $\mathbb{R}^n$ vector mapping of the graph's vertices as input to the SkipGram algorithm [24]. For each available sentence, this method updates the current $\mathbb{R}^n$ vector representations by maximising the co-occurrence probabilities among the vertices ("words" in the vocabulary) that appear within a window of size w. The posterior distribution is learned using Hierarchical Softmax [25].

E. Data Representations to Distance Metrics

Each of the presented graph mining techniques involves computing a matrix representation of a graph, where the individual nodes are expressed as multidimensional vectors – the rows of the output matrix. Because of the imposed known node correspondence constraint, where every graph in a considered population has an equal number of nodes with known IDs, we can obtain a measure of distance between the graphs by processing them with the same algorithm and comparing the resulting matrices. This way of evaluating the change in connectivity between graphs with an identical set of vertices is currently applied in DeltaCon [16]. However, the matrix comparison method that is used, RootED, assumes that all matrix entries are in the $[0, 1]$ range. This is not necessarily the case for other algorithms – e.g., the multidimensional vector embeddings produced by DeepWalk can hold negative values. As such we compare the matrices by measuring the Euclidean or Cosine distance between pairs of rows and aggregating the results (i.e., comparing the different representations for corresponding nodes):

$$Euclidean(M_1, M_2) = \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{n} (M_{1,ij} - M_{2,ij})^2}$$

$$COS(M_1, M_2) = \sum_{i=1}^{n} 1 - \frac{M_{1,i} \cdot M_{2,i}}{||M_{1,i}||\,||M_{2,i}||}$$

V. Experimental Evaluation

To evaluate the performance of the presented distance measures we conducted experiments involving collections of graphs with prior knowledge of their intended cluster assignments. Each of the considered datasets emphasises unique structural distortions amongst the target clusters, showcasing the limitations of different algorithms. A common framework was created in Python, including implementations for GED, DeltaCon, SimRank and SFE, while provided distributions were used for RolX[1] and DeepWalk[2].

A. Execution Environment

All experiments were conducted on a system with two Intel Xeon E5-2660 2.20GHz CPUs and 64GB of RAM running an installation of Ubuntu 14.04. GCC 4.8.4 was used to compile the provided implementation of RolX as part of the Stanford Network Analysis Platform – Release 4.0, while a Python 2.7.6 environment was used for executing the remaining algorithms, which heavily depend on numpy 1.14. The considered release of DeepWalk is 1.0.3.

B. Graph Generator and Data Sets

A custom graph generator was used to build the experimental data sets. To produce a collection of graphs $\mathcal{G}$ the generator requires the size of the collection g, the number of nodes n in each graph (there is a known node correspondence between all graphs in the collection), and a partitioning of the graphs' nodes into k subsets. Each partition p_k of V_i (where V_i is the vertex set of a graph $G_i \in \mathcal{G}$) is a continuous sequence of nodes, defined using a (*start node ID*, *end node*

[1]https://github.com/snap-stanford/snap/tree/master/examples/rolx
[2]https://github.com/phanein/deepwalk

ID) tuple. Additionally, a connectivity parameter is supplied for p_k, indicating how many edges should be placed among its nodes – calculated as a percentage of the edges in a clique of size $|p_k|$. The number of edges between different partitions (i.e., edges (u, v), where $u \in p_k$ and $v \in p_l$, $k \neq l$) can also be adjusted. With this configuration, we can model an output graph as a collection of regions with varying internal and cross-region connectivity.

Ideally, we would expect that graphs with similar structural properties (i.e., same degrees of intra- and cross-region connectivity) are grouped together by the clustering algorithm. Equivalently, we would like to see such similarly structured graphs have a small distance value computed using the distance metrics mentioned above. As such, our test datasets consisted of several sub-collections of graphs, each created using a different set of input parameters for the graph generator (keeping the size of the vertex set n constant as we assume known node correspondence).

The following data sets were generated to evaluate the silhouette performance of the various distance measures:

DS1: A collection of 400 graphs with 20 nodes each, where the vertex set of each graph is split into four equally sized regions: (1 - 5), (6 - 10), (11 -15), and (16 - 20). In the first 100 graphs, region (1 - 5) is set to contain 90% of the edges in a clique of size 5, while the remaining regions contain 20%. For each subsequent 100 graphs, the region connectivity values are shifted clockwise. Cross-region connectivity is adjusted, such that there are multiple edges between the densely connected region and the sparse regions. This effectively creates a series of graphs, with a well-connected core and several other outlier nodes, where the position of the core moves for each of the four groups.

DS2: A collection of 500 graphs with 50 nodes each, where the vertex set of each graph is split into five regions: (1 - 10), (11 - 20), (21, 30), (31, 40), and (41 -50). For each group of 100 graphs, a different permutation of the following region connectivity values is assigned: 80%, 60%, 40%, 20%, and 10%. The cross-region connectivity is set so that there are $5 \times \binom{5}{2}$ edges randomly positioned between the regions. This configuration models a network with semi-isolated communities of nodes, where the position of different community types (densely/sparsely connected) is shifted.

DS3: A collection of 200 graphs with 40 nodes each, where the vertex set of each graph is split into two regions: (1 - 20) and (21 - 40), both with an internal connectivity value of 80%. In the first 100 graphs, the two regions remain completely isolated (i.e., there are 0 cross-region edges), while in the second group, 20 edges are randomly placed between the two regions. This setup expresses the difference between having independent and loosely coupled communities.

To measure the algorithms' runtime and scalability potential we consider an additional 10 data sets, with 5 of them containing 500 graphs each and the rest having 1000. The number nodes per graph $n = |V|$ for the individual data sets in these two groups is also varied, starting from 50 nodes per graphs, and increasing to 100, 200, 500 and 1000 (table I).

C. Evaluation Metrics

A standard way of evaluating the quality of a performed clustering operation is by calculating the Within-Cluster Sum of Squares (WCSS). This refers to the sum of squared distances between each data point and its nearest centroid:

$$WCSS = \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

In the context of comparing distance functions, however, WCSS produces hard to interpret results. This is because the distances between two graphs in a given dataset can vary greatly depending on the choice of dissimilarity measure and these values are not directly comparable without performing normalisation. An alternative approach for comparing the properties of different distance functions in a clustering scenario is to use Silhouette [26]. Silhouette is a method of interpretation and validation of consistency within clusters of data. It offers a succinct way of representing how well each data point lies within its specified cluster. The silhouette score of a particular observation in the input set is a measure of how similar that object is to its own cluster in comparison to other clusters. This value can range from -1 to 1 so that:

- Values near 1 indicate a near perfect assignment – the observation is closely matched to its own cluster and poorly matched to its neighbouring clusters.
- Values near 0 indicate that the assignment is indifferent – e.g., the data point is equidistant from its current and neighbouring cluster.
- Values near -1 indicate an incorrect assignment – the observation is closer to a different cluster from the one its assigned to.

The cluster model as a whole is then considered to be well adjusted to the input data distribution if most points have a high score. The silhouette value for any observation i is computed as:

$$S(i) = \frac{B(i) - A(i)}{max\{A(i), B(i)\}}$$

where A(i) is the average distance from i to all other points in its assigned cluster and B(i) is the lowest average distance from i to the points in any other cluster (the neighbouring cluster of i). The silhouette value for the entire model is, therefore, the mean of the scores of all data items.

While this evaluation method is intended to aid in identifying the optimal value for K it can easily be adapted for comparing different distance measures. By supplying a data set with known groups of similar items, along with their intended cluster labels, we only need to vary the way distances are computed between the individual items. Although the cluster assignments in each run would not change, the values returned by silhouette will differ. In particular, high scores would indicate that the given distance measure "agrees" with the provided cluster labels (i.e., the distance between elements in any single cluster is low, while the distance between elements in different clusters is high) and low scores would mean that the distance measure was unable to detect the

structural dissimilarities between items in different groups. Note that larger silhouette values do not necessarily correspond to improvements in quality – e.g., having a distance of 0 between all items within a cluster may not be desirable. Instead, silhouette provides an indicator of how "confident" the particular distance measure is that the items should be partitioned according to the supplied labels.

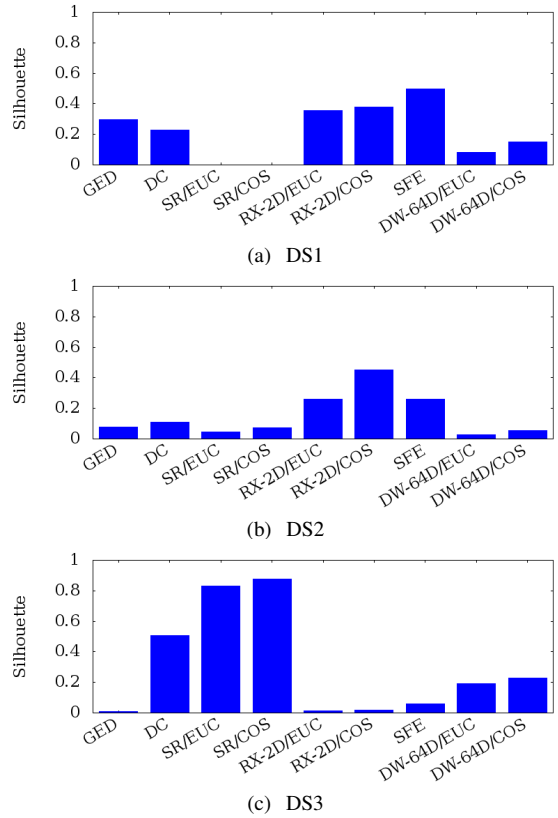
For each of DS1 - 3, we measure the silhouette performance of a chosen distance function by first setting any necessary input parameters and computing a $g \times g$ matrix of dissimilarity values between the considered graphs ($g = |\mathcal{G}|$). A silhouette score is then calculated for the target cluster membership labels, such that all graphs generated with identical regional connectivity parameters are assigned to the same cluster: e.g., in DS1, the first 100 graphs all have a label of 0, the next 100 have a label of 1, etc. The produced value is then a measure of how much the particular distance function "agrees" with the specified partitioning.

Note that while Graph Edit Distance and Deltacon can be applied directly, there is a plethora of available configurations for the outlined graph mining methods. As such, we consider the impact of their adjustable input parameters along with the different ways of comparing their output matrices. In the case of SFE, however, having a disconnected node would result in a division by zero when computing Cosine distance – this constrains SFE to be exclusively used with Euclidean distance.

In regards to evaluating the runtime performance of the individual algorithms, because we are interested in the problem of clustering a collection of graphs, we measure the time necessary to compute the $g \times g$ matrix of dissimilarity values between any two graphs in a given dataset $\mathcal{G}$. This procedure is performed in two separate stages. First, we pre-compute the adjacency matrix for each graph in $\mathcal{G}$ and perform a mapping operation, such that each matrix is individually transformed using the chosen graph mining method – e.g., Fast Belief Propagation in the case of DeltaCon (this phase is not present for GED as it is the only algorithm that directly operates on adjacency matrices). Next, a matrix comparison method is applied to compute the $\binom{g}{2}$ dissimilarity values between all pairs of graph embeddings produced in the previous stage (the $g \times g$ matrix is mirrored with zeros along the diagonal). Finally, we return the aggregated runtime for both jobs.

Because the implementations for RolX and DeepWalk are external to our framework, the way they are used in the outlined mapping stage involves running them as standalone processes, operating on text files that store graph representations. While this approach results in a noticeable performance penalty for small datasets, we show that the added overhead is not impactful for larger workloads.

Once the $g \times g$ matrix is returned, a clustering algorithm can more efficiently be used by performing lookups for the distances between graphs rather than re-evaluating them multiple times. Because of this, we observed that pre-computing the dissimilarity matrix for g graphs takes far longer than the subsequent clustering operation. Although we experimented with multiple clustering algorithms, due to their runtime being negligible in relation to calculating the $g \times g$ matrix (and clearly unaffected by choice of distance function used to obtain the dissimilarity matrix) we do not report on their performance.



(a) DS1

(b) DS2

(c) DS3

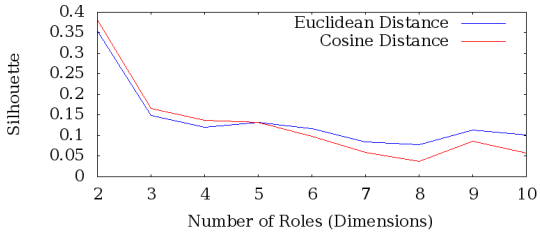
Fig. 1: Silhouette scores for all algorithms (EUC: Using Euclidean distance; COS: Using Cosine distance)

D. Evaluation Results

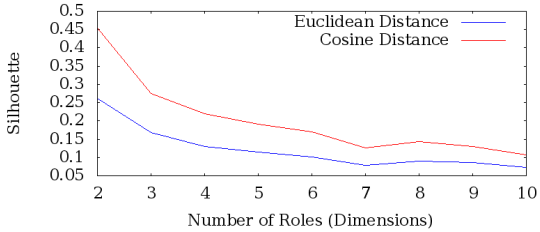
Silhouette Performance: **DS1:** Figure 1a shows the highest Silhouette scores obtained for DS1, using each of the graph distance measures, while figure 3 displays the corresponding heatmaps of dissimilarity values between the considered graphs[3]. Each entry of the presented heatmaps at row i, column j represents the distance between G_i and G_j in DS1, such that darker and brighter colors correspond to lower and higher values respectively (note the black line along the diagonal indicating that the distance from each graph to itself is 0). From these results, it is clear that most of the distance measures were able to detect the positional shift of the densely connected core of nodes among the four groups of graphs.

Graph Edit Distance (fig. 3a) and DeltaCon (fig. 3b) both provide a clean partitioning of the intended graph clusters (the observable squares along the diagonal), resulting in silhouette values between 0.2 and 0.3. SFE, however, manages to obtain the highest score among the considered measures (0.49), despite some perceived noise in its corresponding

[3]The presented heatmaps for SimRank, RolX and DeeplWalk showcase the results of using Cosine distance as means of comparing the algorithms' output matrices, as this method produces higher Silhouette scores and more easily observable clusters.



(a) DS1



(b) DS2

Fig. 2: Silhouette scores for RolX vs number of roles

heatmap (fig. 3e). Because of the comparatively small size of the graphs in DS1, within each cluster, the corresponding nodes will have highly similar degree distributions – giving SFE its discriminative power. In contrast, SimRank (fig. 3c), using either Euclidean or Cosine distance to compare matrix representations, is completely unable to identify the target community structure, with a silhouette score near 0.

For RolX, we examined the effect of changing the number of roles that are computed for each node, along with comparing the output matrices using Euclidean and Cosine distance. Figure 2a shows that by increasing the dimensionality of the produced representation, the algorithm's potential to distinguish between the four clusters is generally reduced. Indeed, the largest silhouette score is obtained when using a $\mathbb{R}^2$ embedding of the graphs' nodes, with Cosine distance (0.38) providing marginally higher value than Euclidean (0.35). While RolX operates on the output of a feature extraction algorithm (ReFeX), this result is still lower than the one computed for SFE. In the corresponding heatmap (fig. 3d) we can observe that graphs within the first group (top left square) are not considered to be part of a cohesive cluster – i.e., the algorithm has identified additional underlying structural differences. As such, the silhouette score for that group is considerably lower, decreasing the overall score for the dataset.

When evaluation DeepWalk, a variety of possible configurations were attempted – changing the dimensionality of the produced embedding, increasing/decreasing the walk length and the number of walks started at each node, using different window sizes for the language modelling phase, etc. In all scenarios, the algorithm was found to have low discriminative power, with the highest result obtained at: 64-dimensional vertex representations, with ten walks of length 50 started at each node and a window size of 5. We also observed that comparing DeepWalk's output matrices using Cosine distance results in a noticeable increase in silhouette score versus Euclidean distance – 0.15 and 0.08 respectively.

DS2: The results obtained for DS2 are presented in fig. 1b and 4. We can see that while the ranking of the individual algorithms has shifted, using methods that rely on feature extraction again provides the most segregated clusters. One observation about the data in DS2 is that the regional connectivity properties for the first two groups are highly similar, resulting in lower distances between the graphs in those clusters and an overall reduction in the computed silhouette scores.

In this scenario, the performance of Graph Edit Distance (0.07) is much lower compared to DS1, because of the increased graph size. As the number of edges in a clique $n \times (n-1)/2$ grows quadratically with the size of the vertex set n, if n is increased, the possibilities for placing a low percentage of those edges amongst the vertices will also rise. Therefore, regardless of the identical parameters used to generate the graphs of a target cluster, GED will return high dissimilarity values.

DeltaCon outperforms GED, due to its ability to identify the shift in regional connectivity values between the different clusters – i.e., all nodes within a densely linked region will have high affinities with each other, regardless of the exact placement of the edges. This property is also present in SimRank, which is now able to identify the intended community structure, although its discriminative power is still low.

SFE once again produces a large silhouette value (0.26), as a result of the similar degree distribution of the corresponding nodes within a cluster. RolX, however, manages to obtain the highest score (0.45) when using a $\mathbb{R}^2$ representation for each vertex and Cosine distance. Similar to the behaviour observed for DS1, figure 2b shows that assigning more roles to each vertex diminishes the algorithm's ability to differentiate between graphs in different clusters. Although, in this scenario, the choice of matrix comparisons method is found to have a greater impact on silhouette, with Cosine distance providing larger values than Euclidean.

Finally, applying DeepWalk results in the lowest score (0.05) for DS2 – i.e., the least amount of segregation between the target clusters. This result was obtained with the same parameters used for DS1, with alternative configurations not providing any improvement.

DS3: Figures 1c and 5 present the gathered results for DS3, which indicate a new trend in the silhouette performance of the analyses distance measures. In particular, because of the unique difference among the graphs in the two clusters of DS3 (graphs in the first cluster have two densely connected regions with no edges between them, while in the second cluster all graphs have 20 edges linking nodes in the two regions), the observed values reflect the sensitivity of each algorithm to disconnected components.

DeltaCon and SimRank, both being methods that rely on the spread of node affinities throughout a graph, can find a clear separation between the intended clusters. Because they produce an $\mathbb{R}^n$ representation for each vertex ($n = |V|$) when there is no link between two sets of vertices, all of the values referring to nodes in the opposite region will be 0. In the examined graphs, the two separate regions both contain 50%

of the vertex set, and so their separation would have the highest impact on the propagated affinities.

Unlike the results for DS1 and DS2, SFE and RolX both produce silhouette values near 0. This is expected for SFE, as the inclusion of the additional cross-region edges does not greatly alter the degree distribution of corresponding nodes. For RolX, the obtained vertex representations could not express the behavioural function of nodes, acting as the connecting points between two dense regions, regardless of the number of specified roles. Graph Edit Distance also provides a low score, as the added dissimilarity from the presence of the cross-region edges is far smaller than that of the variance in edge placement within the individual regions.

DeepWalk returns an average scores (0.23), as it considers the graphs within the first group to be highly similar (fig. 5f) – walks started in one region cannot transition to the nodes of another, and so each of the "sentences" used in the language modeling phase contain elements from one half of the vocabulary. As such, the deep learning method builds an association only between vertices in the same region. The second group of 100 graphs are not identified as a coherent cluster, however, due to the random positioning of the cross-region edges – the random walks move between regions at different locations, resulting in inconsistent models.

Runtime Evaluation: Table I displays the performance of each distance measure when operating on larger datasets. The presented values show the aggregated runtime for executing a mapping stage, where the relevant graph mining algorithm is used to produce a matrix representation for each of the available graphs, along with a matrix comparisons stage, where the $\binom{g}{2}$ distances between any two matrices are calculated ($g = |\mathcal{G}|$). For SimRank, RolX and DeepWalk we measure the impact of using either Euclidean or Cosine distance for the second stage. Additionally, when benchmarking RolX and DeepWalk, we only consider the set of input parameters that were shown to provide the largest silhouette values in the previous experiment. As such, we apply RolX with the number of extracted roles per vertex set to 2 (creating higher dimensional representations has a greater computational cost), and for DeepWalk, we use an $\mathbb{R}^{64}$ vertex representation, with ten walks of length 50 started at each node and a window size of 5. Note that while the process of performing random walks through a graph can be parallelised using multiple workers, the scope of this evaluation is limited to measuring single-threaded performance.

From the obtained results we can see that SFE, due to its linear complexity and low dimensional representation for each vertex, outperforms all other algorithms. Graph Edit Distance is second at runtime, despite being the only method that does not require a mapping stage (it operates directly on the adjacency matrices). This is because each of the $\binom{g}{2}$ comparisons that it performs involves $n \times n$ matrices (where $n = |V|$), while SFE operates on the $n \times 2$ representations, produced in its mapping stage. DeltaCon and SimRank are next, as they both include an expensive mapping stage and subsequent comparisons between $n \times n$ matrices (each row holds the computed affinity between node i and all other nodes in the graph). For SimRank, we also observe that using Cosine distance after the mapping stage results in far lower performance than Euclidean.

In the case of RolX and DeepWalk, because the resulting vertex representations have few dimensions, the execution time is heavily dominated by the mapping phase – transitioning from 500 to 1000 graphs results in a $2\times$ slow down. This observation also applies to SFE, as the output matrices of all three algorithms grow linearly with the size of the vertex set. In contrast, the matrices used by GED, SimRank and DeltaCon grow quadratically – also heavily affecting the comparison stage. As such, when performing a mapping with RolX or DeepWalk, the differences between using Euclidean and Cosine distance are far smaller than for SimRank.

VI. Conclusions

In this paper, we surveyed a range of existing graph similarity measurement methods and proposed several new ones based on graph mining algorithms designed for node classification tasks or community discovery within a network, and investigated their performance in the context of measuring distances between graphs with known node correspondence. Furthermore, we presented a novel approach for using the silhouette algorithm, to quantify how much a particular distance function "agrees" with a specified cluster membership assignment; this method was used alongside custom generated data sets, with known clusters of graphs with similar connectivity properties, to determine the sensitivity of each distance measure to unique structural distortions. Our performance evaluation shows that there is no single distance measure that returns the highest silhouette score in all scenarios, making the choice dependent on the characteristics of the considered graph collection and the available compute resources. Last, we discussed the applicability of each distance metric for various characteristics of the graphs in the dataset.

References

[1] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, Jan 2008.

[2] A. Singhal *et al.*, "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001.

[3] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.

[4] I. Popescu, K. Portelli, C. Anagnostopoulos, and N. Ntarmos, "The case for graph-based recommendations," in *Proc. IEEE Big Data*, Dec 2017, pp. 4819–4821.

[5] Semantic Technologies Laboratory, "Open Ontology Repository," http://oor.net/.

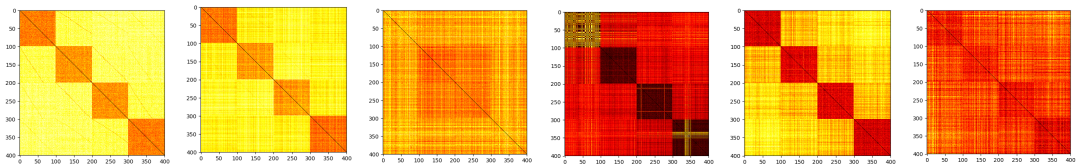
[6] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. ACM SIGKDD KDD*, 2005.

[7] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. ACM STOC*, 1971, pp. 151–158.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

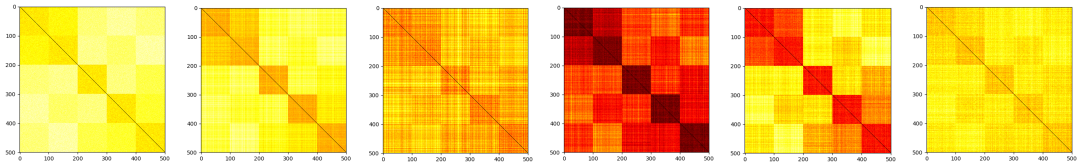
[9] R. Hoffmann, C. McCreesh, and C. Reilly, "Between subgraph isomorphism and maximum common subgraph," in *Proc. AAAI*, 2017.

[10] A. Sanfeliu and K. S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 3, pp. 353–362, May 1983.



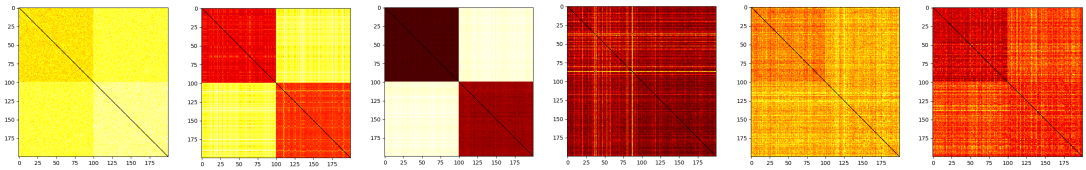
(a) GED (b) DC (c) SR (d) RX 2D (e) SFE (f) DW 64D

Fig. 3: Heatmaps of the distance values between any two graphs in DS1



(a) GED (b) DC (c) SR (d) RX 2D (e) SFE (f) DW 64D

Fig. 4: Heatmaps of the distance values between any two graphs in DS2



(a) GED (b) DC (c) SR (d) RX 2D (e) SFE (f) DW 64D

Fig. 5: Heatmaps of the distance values between any two graphs in DS3

Data Set	**GED**	**DC**	**SFE**	**SR ED**	**SR COS**	**RX 2D ED**	**RX 2D COS**	**DW 64D ED**	**DW 64D COS**
$\lvert\mathcal{G}\rvert$ = 500, $\lvert V\rvert = 50$	3.68	7.56	1.79	2.98	14.11	21.71	31.03	760.51	672.75
$\lvert\mathcal{G}\rvert$ = 500, $\lvert V\rvert = 100$	6.92	18.89	1.85	5.36	22.96	32.31	50.63	1068.26	943.61
$\lvert\mathcal{G}\rvert$ = 500, $\lvert V\rvert = 200$	22.18	66.83	1.92	19.97	60.58	75.36	86.14	1909.49	1842.91
$\lvert\mathcal{G}\rvert$ = 500, $\lvert V\rvert = 500$	283.83	1414.99	3.19	130.23	491.22	364.36	352.52	6175.61	6216.07
$\lvert\mathcal{G}\rvert$ = 500, $\lvert V\rvert = 1000$	796.57	1691.69	3.33	862.12	1527.14	1187.87	1225.21	13021.58	13181.59
$\lvert\mathcal{G}\rvert$ = 1000, $\lvert V\rvert = 50$	12.28	25.07	4.48	8.23	40.44	27.82	70.75	1249.91	1904.21
$\lvert\mathcal{G}\rvert$ = 1000, $\lvert V\rvert = 100$	28.34	74.74	7.32	18.73	94.31	58.92	130.78	1861.43	1827.15
$\lvert\mathcal{G}\rvert$ = 1000, $\lvert V\rvert = 200$	72.94	225.33	7.93	70.96	216.88	151.5	177.04	3694.94	3587.08
$\lvert\mathcal{G}\rvert$ = 1000, $\lvert V\rvert = 500$	422.13	1352.52	9.76	565.82	1207.57	931.36	966.43	11841.43	11977.04
$\lvert\mathcal{G}\rvert$ = 1000, $\lvert V\rvert = 1000$	2162.11	6461.86	15.21	1894.31	8437.65	3571.48	3647.61	26217.87	26449.32

TABLE I: Runtime, in seconds, for computing the distances between any two graphs in a data set.

[11] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974.

[12] S. M. Selkow, "The tree-to-tree editing problem," *Information processing letters*, vol. 6, no. 6, pp. 184–186, 1977.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., November 1999.

[14] J. M. Kleinberg, "Hubs, authorities, and communities," *ACM Comput. Surv.*, vol. 31, no. 4es, Dec. 1999.

[15] G. Jeh and J. Widom, "Simrank: A measure of structural-context similarity," in *Proc. ACM SIGKDD KDD*, 2002, pp. 538–543.

[16] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "Deltacon: A principled massive-graph similarity function," in *Proc. SIAM Intl. Conf. on Data Mining*, 2013, pp. 162–170.

[17] D. Koutra *et al.*, "Unifying guilt-by-association approaches: Theorems and fast algorithms," in *Proc. ECML/PKDD*, 2011, pp. 245–260.

[18] H. Tong, C. Faloutsos, and J.-Y. Pan, "Random walk with restart: fast solutions and applications," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 327–346, 2008.

[19] V. Hautamaki, I. Karkkainen, and P. Franti, "Outlier detection using k-nearest neighbour graph," in *Proc. ICPR*, vol. 3, Aug 2004, pp. 430–433.

[20] K. Henderson *et al.*, "Rolx: Structural role extraction & mining in large graphs," in *Proc. ACM SIGKDD KDD*, 2012, pp. 1231–1239.

[21] ——, "It's who you know: Graph mining using recursive structural features," in *Proc. ACM SIGKDD KDD*, 2011, pp. 663–671.

[22] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, Sep. 1978.

[23] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. ACM SIGKDD KDD*, 2014, pp. 701–710.

[24] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[25] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proc. Aistats*, 2005, pp. 246–252.

[26] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.