Leveraging Graph Dimensions in Online Graph Search

Yuanyuan Zhu
State Key Laboratory of Software Engineering
Wuhan University, China
yyzhu@whu.edu.cn

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hong Kong, China
yu@se.cuhk.edu.hk

Lu Qin
Centre for QCIS, FEIT,
University of Technology, Sydney, Australia
lu.qin@uts.edu.au

ABSTRACT

Graphs have been widely used due to its expressive power to model complicated relationships. However, given a graph database $\mathcal{D}_\mathcal{G} = \{g_1, g_2, \cdots, g_n\}$, it is challenging to process graph queries since a basic graph query usually involves costly graph operations such as maximum common subgraph and graph edit distance computation, which are NP-hard. In this paper, we study a novel *DS-preserved* mapping which maps graphs in a graph database $\mathcal{D}_\mathcal{G}$ onto a multidimensional space $\mathcal{M}_\mathcal{G}$ under a structural dimension $\mathcal{M}$ using a mapping function $\phi()$. The *DS-preserved* mapping preserves two things: distance and structure. By the distance-preserving, it means that any two graphs g_i and g_j in $\mathcal{D}_\mathcal{G}$ must map to two data objects $\phi(g_i)$ and $\phi(g_j)$ in $\mathcal{M}_\mathcal{G}$, such that the distance, $d(\phi(g_i), \phi(g_j))$, between $\phi(g_i)$ and $\phi(g_j)$ in $\mathcal{M}_\mathcal{G}$ approximates the graph dissimilarity $\delta(g_i, g_j)$ in $\mathcal{D}_\mathcal{G}$. By the structure-preserving, it further means that for a given unseen query graph q, the distance between q and any graph g_i in $\mathcal{D}_\mathcal{G}$ needs to be preserved such that $\delta(q, g_i) \approx d(\phi(q), \phi(g_i))$. We discuss the rationality of using graph dimension $\mathcal{M}$ for online graph processing, and show how to identify a small set of subgraphs to form $\mathcal{M}$ efficiently. We propose an iterative algorithm DSPM to compute the graph dimension, and discuss its optimization techniques. We also give an approximate algorithm DSPMap in order to handle a large graph database. We conduct extensive performance studies on both real and synthetic datasets to evaluate the top-k similarity query which is to find top-k similar graphs from $\mathcal{D}_\mathcal{G}$ for a query graph, and show the effectiveness and efficiency of our approaches.

1. INTRODUCTION

A graph models complex structural relationships among objects and has been extensively used in a wide range of applications, such as chemical compound structures in chemistry, attributed graphs in image processing, food chains in ecology, electrical circuits in electricity, protein interaction networks in biology, etc. With the increasing popularity of graph databases that contain a number of graphs in various applications, graph manipulations become very important as they are useful in most of the common knowledge discovery tasks like classification, clustering, outlier detection, graph indexing, etc.

Proceedings of the VLDB Endowment, Vol. 8, No. 1

Despite the great expressive power of graphs, the graph query processing in real applications is very challenging, because the basic graph operations are very costly, such as maximum common subgraph and graph edit distance, which are NP-hard. Such high complexity causes a severe consequence, i.e., a lack of efficient algorithms and tools to process and analyze the graphs. The key issue we address in this paper is how to achieve high quality and efficiency in online graph query processing by mapping graphs onto a multidimensional space. This approach is taken in the PubChem Compound Database (http://pubchem.ncbi.nlm.nih.gov) for users to search similar graphs for a given query graph. For the PubChem Compound Database, the domain experts have identified a dictionary-based binary fingerprint in total 881 dimensions, where the dimensions correspond to the substructures as well as some other additional information. Such dimensions are very difficult even for domain experts to identify, and the process of identifying such dimensions is very time consuming and expensive, due to the large number of combinations of possible features in the graph database to analyze, which can take months or even years. Motivated by this, in this paper, we study how to automatically identify a small set of subgraphs to form a graph dimension $\mathcal{M}$, for effective and efficient graph processing in a graph database $\mathcal{D}_\mathcal{G}$.

The main contributions of this paper are summarized below. First, we study a new *DS-preserved* mapping which maps graphs in a graph database $\mathcal{D}_\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ onto a multidimensional space $\mathcal{M}_\mathcal{G}$ under a structural graph dimension $\mathcal{M}$ using a mapping function $\phi()$, where each dimension represents a feature (or subgraph) of graphs in $\mathcal{D}_\mathcal{G}$. The *DS-preserved* mapping preserves two things: distance and structure. By the distance-preserving, it means that any two graphs g_i and g_j in $\mathcal{D}_\mathcal{G}$, must map to two data objects $\phi(g_i)$ and $\phi(g_j)$ in $\mathcal{M}_\mathcal{G}$, such that the distance between $\phi(g_i)$ and $\phi(g_j)$ in $\mathcal{M}_\mathcal{G}$, $d(\phi(g_i), \phi(g_j))$, approximates the graph dissimilarity $\delta(g_i, g_j)$ in $\mathcal{D}_\mathcal{G}$. By the structure-preserving, it further means that for a given unseen query graph q, which does not necessarily appear in $\mathcal{D}_\mathcal{G}$, the distance between q and any graph g_i in $\mathcal{D}_\mathcal{G}$ needs to be preserved such that $\delta(q, g_i) \approx d(\phi(q), \phi(g_i))$. Second, we show the rationality of using the graph dimension $\mathcal{M}$ for online graph processing by deriving a bound which indicates that the structure-preserving can be maintained if distance-preserving is achieved. Third, we show how to identify a small set of subgraphs to form $\mathcal{M}$ efficiently. We propose an iterative algorithm DSPM, and discuss its optimization techniques. We also give an approximate algorithm DSPMap in order to handle a large graph database. Fourth, we conduct extensive performance studies using both real and synthetic datasets to evaluate top-k similarity query which is to find top-k similar graphs from $\mathcal{D}_\mathcal{G}$ for a query graph, and show the effectiveness and efficiency of our approaches.

The rest of the paper is organized as follows. Section 2 gives the

problem statement. Section 3 reviews the related work. Section 4 discusses *DS-preserved* mapping. In particular, we discuss the rationality of using the graph dimension and the model for dimension computation. Section 5 discusses the algorithms. We give an iterative algorithm, discuss its optimization techniques, and also give an approximate algorithm. Section 6 shows the effectiveness and efficiency of our algorithms with extensive experimental studies. Section 7 concludes this paper.

2. PROBLEM STATEMENT

In this paper we deal with undirected labeled graphs. Given a set of labels Σ, an undirected labeled graph is represented as $g = (V, E, l)$, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, and l is a labeling function on vertices and edges. We use $V(g)$ and $E(g)$ to denote the vertex set and edge set of graph g, respectively, and use $|V(g)|$ and $|E(g)|$ to denote the number of vertices and the number of edges in g.

Let $\mathcal{D}_\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ be a graph database where every g_i is an undirected labeled graph. The graph query processing is to obtain results, denoted as $\mathcal{D}_\mathcal{G}(q)$, from $\mathcal{D}_\mathcal{G}$, for a user given query graph q. The cost of such graph query processing is high because it involves costly graph operations, such as maximum common subgraph, graph edit distance, etc. In order to significantly improve the efficiency while maintaining the high quality of the query results, the key is to explore the possibility of replacing the costly graph operations with much cheaper operations on a multidimensional space. In brief, given a graph database $\mathcal{D}_\mathcal{G}$, let $\mathcal{D}_\mathcal{G}(q)$ denote the query processing result for a given query graph q. Instead of conducting query processing using high cost graph operations, we map the graphs in $\mathcal{D}_\mathcal{G}$ and any unseen query graph to a multidimensional database $\mathcal{D}_\mathcal{M}$ using a mapping function $\phi()$, and obtain the results $\mathcal{D}_\mathcal{M}(\phi(q))$ ($\approx \mathcal{D}_\mathcal{G}(q)$). The challenging issue is on the mapping. Let $\delta(g_i, g_j)$ denote a graph dissimilarity function between two graphs g_i and g_j in $\mathcal{D}_\mathcal{G}$, and let $d(\phi(g_i), \phi(g_j))$ denote a distance function between two data objects $\phi(g_i)$ and $\phi(g_j)$ in $\mathcal{M}_\mathcal{G}$ that two graphs map to. The mapping needs to preserve two things: distance and structure. We call it *DS-preserved* mapping. By the distance-preserving, it means that any two graphs g_i and g_j in $\mathcal{D}_\mathcal{G}$, must map to two data objects $\phi(g_i)$ and $\phi(g_j)$ in $\mathcal{M}_\mathcal{G}$, such that $\delta(g_i, g_j) \approx d(\phi(g_i), \phi(g_j))$. By the structure-preserving, it further means that for a given unseen query graph q, which does not necessarily appear in $\mathcal{D}_\mathcal{G}$, the distance between q and any graph g_i in $\mathcal{D}_\mathcal{G}$ needs to be preserved such that $\delta(q, g_i) \approx d(\phi(q), \phi(g_i))$. It is important to note that by structure-preserving, it requests to keep the most informative fingerprint of the graphs in the entire $\mathcal{D}_\mathcal{G}$.

In this paper, for $\delta(g_i, g_i)$, we focus on two graph dissimilarities [1] [2] based on Maximum Common Subgraph (*MCS*). A graph g is a *MCS* of two graphs g_i and g_j, denoted as $\mathsf{mcs}(g_i, g_j)$, if g is a common subgraph of g_i and g_j and there is no other common subgraph g' larger than g. Note that a graph g' is a subgraph of g if g' is subgraph isomorphic to g. Here we can also say that g is a supergraph of g' or g contains g'. We denote such relationship as $g' \subseteq g$ or $g \supseteq g'$.

The first graph dissimilarity [1] is defined below.

$$\delta_1(q, g) = 1 - \frac{|E(\mathsf{mcs}(q, g))|}{\max\{|E(q)|, |E(g)|\}} \quad (1)$$

The second graph dissimilarity [2] is defined as follows.

$$\delta_2(q, g) = 1 - \frac{2|E(\mathsf{mcs}(q, g))|}{|E(q)| + |E(g)|} \quad (2)$$

These two dissimilarities measure the graph structures from different points of view. The former is normalized by the maximum graph size, which emphasizes the difference between the maximum common graph and the larger graph. The latter is normalized by the average graph size, which emphasizes the difference between the maximum common graph and both graphs.

Below, we use δ to indicate both δ_1 and δ_2 unless otherwise specified, and use δ_{ij} for $\delta(g_i, g_j)$. The graph dissimilarity $\delta(,)$ is symmetric and is in the range of $[0, 1]$.

Problem Statement: Given a graph database $\mathcal{D}_\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ and a graph dissimilarity function $\delta(,)$, identify a structural dimension $\mathcal{M}$ with a small number of subgraphs such that it can *DS-preserved* map graphs (including those graphs in $\mathcal{D}_\mathcal{G}$ as well as any unseen query graph) onto the multidimensional space $\mathcal{M}_\mathcal{G}$ regarding an Euclidean distance function $d(,)$ defined on $\mathcal{M}_\mathcal{G}$. As a result, a multidimensional database $\mathcal{D}_\mathcal{M}$ is constructed that consists of $\phi(g_i)$ for any $g_i \in \mathcal{D}_\mathcal{G}$, and it will be used to process a query $\phi(q)$ for a query graph q. In this paper, we use top-k query for graph query processing, which finds the top-k similar graphs in $\mathcal{D}_\mathcal{G}$ for a given query graph. Nevertheless, the identified structural dimension $\mathcal{M}$ can also be applied in many other graph applications such as graph pattern matching and graph clustering.

3. RELATED WORK

Our work is geared towards mapping graphs to a multidimensional space to preserve the distance and structure simultaneously. In this section, we briefly review the five related topics of our work, i.e., graph embedding, graph kernels, feature selection, frequent subgraph mining, and frequent subgraph based indexing.

Graph embedding. Existing work for graph embedding can be divided into two categories. The first category aims to represent each vertex in a single graph as a vector that best characterizes the similarities/weights between vertex pairs. The vector representation for the vertices can be obtained based on the graph spectral theory [3, 4]. The graph spectral techniques are also adopted for dimensionality reduction in multidimensional space. Representative works such as Isomap [5], Locally Linear Embedding [6], and Laplacian Embedding [7], can all be interpreted in a general graph embedding framework with different choices of the graph structures. Tetsuo et al. [8] study the trade-off between time and space of graph embedding. Note that approaches in this category aim to transform the vertices of a single graph but not a collection of graphs to vectors, thus inapplicable to the problem studied in this paper.

The approaches in the second category aim at representing each graph in a dataset as a feature vector based on graph operations or statistics. Riesen et al. [9] propose a general approach of mapping graphs to multidimensional vectors. They heuristically select k graphs in the graph set as prototypes, and then map each graph to a k-dimensional vector in which the elements represent the graph edit distances between this graph and the prototypes. To improve the quality of the prototypes, they subsequently propose another approach [10] to first use all the graphs in the graph set as prototypes, and then apply feature selection algorithms to eliminate redundant prototypes and reduce the dimensionality. An obvious disadvantage of these two approaches is that they need to perform k times of the costly graph edit distance computation to obtain the k-dimensional vector for a query graph, which does not essentially reduce the the computation complexity in query processing. Gibert et al. [11] propose another embedding methodology to map graphs to vectors based on statistics of the node/edge attributes. However, such method only preserves very little structure information of graphs and cannot be qualified *DS-preserved* mapping.

Graph kernels. Graph kernels aim at computing similarity scores between graphs in a dataset and have been defined on various graph patterns which generally fall into three classes. The first class,

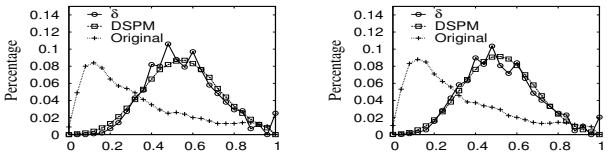
based on random walks/paths, computes the number of matching pairs of random walks/paths in two graphs [12, 13]. The second class, subtree graph kernel [14], has higher representation power of graph structure than the first class, but the processing time grows exponentially while the recursion depth of the subtree patterns becomes deeper. The third class of graph kernels is based on restricted subgraphs, such as Graphlets [15] and h-hop neighbors [16]. However, the Graphlet is only be feasible on unlabeled graphs, and both Graphlets and h-hop neighbors have very limited power to capture the topological structure of graphs as types and sizes of the kernel substructures are very limited. There are also some research on selecting useful features for graph kernels in the literature [17, 18]. However, these approaches aim at achieving higher classification accuracy but not for *DS-preserved* mapping. Thus they are not applicable for the problem studied in this paper.

Feature selection. We review the unsupervised feature selection approaches since this paper focuses on a graph database with no class labels. Exiting unsupervised feature selection methods fall into two categories: wrapper model and filter model [19]. In wrapper approaches, feature selection is wrapped in a specific learning algorithm, which usually results in high computational complexity and less generality so that the selected features are inapplicable to other learning algorithms. So we mainly focus on the filter methods which utilize some intrinsic properties of the data to decide which features should be kept.

Finding the optimal feature subset is intractable [20]. Therefore, greedy search strategies such as sequential forward selection (SFS) [21] and sequential backward elimination (SBE) [20] are typically used. However, these greedy algorithms perform poorly when the feature evaluation criterion is non-monotonic. Meanwhile, various feature evaluation criteria are proposed to evaluate the quality of a subset. For example, Talavera [22] selects features based on feature dependence; Dash et al. [23] choose features based on the entropy of distances between data points; and Mitra et al. [24] select features based on a new feature similarity measure called maximum information compression index (MICI). In recent years, spectral methods are also explored for feature selection, such as [25] which selects features based on the Laplacian score, and a unified framework SPEC [26] which considers Laplacian score as a special case. However, the features selected by these two methods are highly redundant as the correlation among features is neglected. Therefore, a two step approach, MCFS [27], is proposed to find the subset of features instead of evaluating each feature independently. Yang. et al. [28] propose a framework, UDFS, by integrating discriminative information and $\ell_{2,1}$ minimization into one step. A more general framework NDFS [29] is developed to learn the cluster label and feature selection simultaneously where cluster indicator is constrained to be nonnegative. However, they only select the most informative features and do not consider the graph dissimilarity when they are applied in the graph database. Therefore, they cannot achieve good performance on distance-preserving in graph databases, which will also be showed in the experiments later.

Frequent subgraph mining. Frequent subgraph mining has been widely studied in the literature, with the aim of finding the set of the frequent subgraphs, maximal frequent subgraphs, closed frequent subgraphs, and representative frequent subgraphs. A comprehensive survey can be found in [30]. However, the set of frequent subgraphs cannot be used directly as dimensions for *DS-preserved* mapping due to the anti-monotone property of frequent subgraphs.

Frequent subgraph based indexing. In the literature, frequent subgraphs have been used to represent graphs as feature vectors to support graph query, including subgraph containment query and supergraph containment query. To accelerate subgraph containment query process, gIndex [31] and FG-Index [32] are proposed for filtering. gIndex generates all size-bounded frequent subgraphs as well as a subset of size-bounded infrequent subgraphs, while FG-Index indexes all frequent subgraphs with size-increasing support function and additionally includes all infrequent edges. Based on the indexing features used in gIndex, Grafil [33] is developed to support efficient subgraph similarity containment query. To solve the supergraph containment query problem, Chen et al. propose a contrast subgraph-based indexing framework, cIndex [34], to sort out significant and distinctive contrast subgraphs using a redundancy-aware feature selection process. However, the selected frequent subgraphs in the above approaches are only used for efficient filtering but are not aimed to preserve the distance property to avoid the costly verification phase as we study in this paper.



(a) Distribution in $\mathcal{D}_\mathcal{G}$ (b) Distribution between q and $\mathcal{D}_\mathcal{G}$

Figure 1: Dissmilarity/Distance Distribution

4. DS-PRESERVED MAPPING

We study how to identify a multidimensional space $\mathcal{M}$ where each dimension represents a feature (subgraph) taken from $\mathcal{D}_\mathcal{G}$, and map graphs (including $g_i \in \mathcal{D}_\mathcal{G}$ and unseen query graphs) to $\mathcal{M}$ using a mapping function $\phi()$. Below, we use $\mathcal{F}$ instead of $\mathcal{M}$ to indicate that the space is based on features of graphs. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_p\}$, and assume it is identified. A graph g_i can be mapped to a binary vector $\phi(g_i) = \mathbf{y}_i = (y_{i1}, y_{i2}, \ldots, y_{ip})$ where $y_{ir} = 1$ if f_r is a subgraph of g_i, and $y_{ir} = 0$ otherwise. In this paper, we use the normalized Euclidean distance in the mapped space. The Euclidean distance between $\mathbf{y}_i$ and $\mathbf{y}_j$ is defined as $d(\mathbf{y}_i, \mathbf{y}_j) = \left(\frac{1}{p}\Sigma_{r=1}^{p}(y_{ir} - y_{jr})^2\right)^{\frac{1}{2}}$, where $0 \leq d(\mathbf{y}_i, \mathbf{y}_j) \leq 1$.

In order to achieve high quality *DS-preserved* mapping from $\mathcal{D}_\mathcal{G}$ to $\mathcal{M}_\mathcal{G}$, the first difficulty is to discover what properties the mapped space $\mathcal{F}$ should have, and the second difficulty is to identify $\mathcal{F}$ efficiently, considering that the number of possible subgraphs we need to explore is too large for a large graph database $\mathcal{D}_\mathcal{G}$.

A possible way is to restrict the search to only frequent subgraphs. Recall that a subgraph f is frequent if $freq(f) \geq \tau$ where $freq(f) = |sup(f)|/|\mathcal{D}_\mathcal{G}|$ for $sup(f) = \{g_i \mid f \subseteq g_i, g_i \in \mathcal{D}_\mathcal{G}\}$ and $\tau \in [0, 1]$ is a user specified threshold. Such restriction is reasonable, because infrequent subgraphs usually preserve very little structure information of the graphs as they barely occur in $\mathcal{D}_\mathcal{G}$.

However, the set of frequent subgraphs cannot be used directly as an effective multidimensional space for *DS-preserved* mapping due to the anti-monotone property of frequent subgraphs. This property implies that all the subgraphs of a frequent subgraph are also frequent. In brief, given such anti-monotone property, a graph g_i or a query graph q will be mapped to those dimensions that represent the frequent subgraphs contained in g_i or q. This leads to the result that the multidimensional space can be severely unbalanced. Thus, in this paper, a subset of frequent subgraphs are selected to represent the multidimensional space. The frequent subgraph selection is performed by our proposed algorithm, DSPM, to achieve high quality *DS-preserved* mapping.

In Fig. 1, we show the distance/dissimilarity distribution on real chemical compound dataset (see Section 6 for the detailed description of the dataset) for δ, DSPM, Original, where δ is the graph

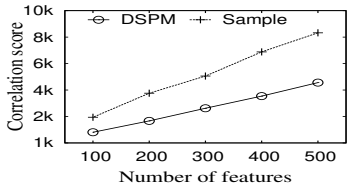


Figure 2: Correlation Score Between Selected Features

dissimilarity based on Eq.(2), DSPM represents the Euclidean distance between data objects in our mapped space, and Original represents the Euclidean distance in the space where all frequent subgraphs are considered as the dimensions. Fig. 1(a) shows the dissimilarity/distance between all graph pairs in $1,000$ randomly selected graphs in the graph database. The x-axis represents the value of the graph dissimilarity/distance, and the y-axis represents the percentage of the graph pairs for each dissimilarity/distance value. We can observe that Euclidean distance generated by DSPM can approximate graph dissimilarity very well while Original cannot, which shows the distance-preserving property of DSPM. We also plot the distribution of the dissimilarity/distance of all graph pairs between $1,000$ randomly selected query graphs and $1,000$ randomly selected database graphs in Fig. 1(b). DSPM can also approximate graph dissimilarity well, which shows the structure-preserving ability of DSPM. These two figures show the necessity of selecting appropriate features for *DS-preserved* mapping.

Fig. 2 shows the sum of the correlation scores between selected features for DSPM and Sample when we vary the dimension number p from 100 to 500 in the real chemical compound dataset, where DSPM is our proposed algorithm and Sample is the algorithm which randomly selects p frequent subgraphs as dimensions. The correlation score between two features is an important measure to evaluate how similar the features are, which is defined using Jaccard Coefficient [35]. As shown in Fig. 2, DSPM has much smaller feature correlation score than Sample, while the query precision of DSPM is twice larger than that of Sample (see Exp-1 in Section 6 for details). Such a result indicates that a better *DS-preserved* mapping tends to use less correlated features so that more structure information of the database graphs can be preserved.

4.1 The Rationality

Recall that *DS-preserved* mapping preserves both distance and structure. For the distance-preserving, it implies $d(\phi(g_i), \phi(g_j)) \approx \delta(g_i, g_j)$, when g_i and g_j in $\mathcal{D}_\mathcal{G}$ are mapped onto $\mathcal{M}_\mathcal{G}$, using $\phi(g_i)$ and $\phi(g_j)$. This can be possibly achieved by identifying $\mathcal{F}$ (a subset of frequent subgraphs) from $\mathcal{D}_\mathcal{G}$, since the number of graphs in $\mathcal{D}_\mathcal{G}$ is fixed. For the structure-preserving, it implies $d(\phi(q), \phi(g_i)) \approx \delta(q, g_i)$ for any graph q that does not appear in $\mathcal{D}_\mathcal{G}$ and any g_i in $\mathcal{D}_\mathcal{G}$, when q and g_i are mapped onto $\mathcal{M}_\mathcal{G}$. This is hard to achieve because the number of possible graph queries is infinite to enumerate.

Below, we assume the distance-preserving can be achieved, and discuss the bound for the structure-preserving in this subsection. Specifically, consider q to be a graph in $\mathcal{D}_\mathcal{G}$ whose mapping quality is bounded, we start our discussion on how to bound the quality for a query graph $q' \subseteq q$ or $q' \supseteq q$. In the following, we show this can be bounded in Corollary 4.1 and Corollary 4.2. To obtain the two corollaries, we first give Lemma 4.1 to show that q' and q, for $q' \subseteq q$, can be bounded by their graph sizes, based on the difference of *MCS* graph similarity between q'/q and a common graph g. With the common graph g as a basis, we prove $\delta(q', g)$ can be bounded by $\delta(q, g)$ for δ_1 and δ_2 in Theorem 4.1 and Theorem 4.2, respectively. We further prove that the mapped distance of q' and g can be bounded by the mapped distance of q and g in

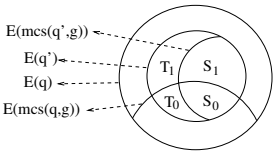


Figure 3: Proof of Lemma 4.1

Theorem 4.3. As a result, the quality of an arbitrary query graph q' can be bounded by Corollary 4.1 and Corollary 4.2. In other words, we bound the quality of $\mathsf{mcs}(q', q)$ by Corollary 4.1 since $\mathsf{mcs}(q', q) \subseteq q$, and bound the quality of q' by Corollary 4.2 since $q' \supseteq \mathsf{mcs}(q', q)$. Given the bound for distance-preserving discussed in this subsection, we discuss how to address the distance-preserving which implies the quality of the structure-preserving in Section 4.2.

Lemma 4.1: *Given a graph g, for a graph q and its subgraph $q' \subseteq q$, the difference between their MCSs with g, $\xi = |E(\mathsf{mcs}(q, g))| - |E(\mathsf{mcs}(q', g))|$, can be bounded as*

$$0 \leq \xi \leq |E(q)| - |E(q')|$$

Proof: We first prove that $\xi \geq 0$. Suppose that $\xi < 0$, which implies that $|E(\mathsf{mcs}(q, g))| < |E(\mathsf{mcs}(q', g))|$. Since $q' \subset q$, we have $\mathsf{mcs}(q', g) \subseteq q' \subseteq q$. Thus we conclude that $\mathsf{mcs}(q', g)$ is a larger common subgraph between q and g than $\mathsf{mcs}(q, g)$, which contradicts the fact that $\mathsf{mcs}(q, g)$ is the *MCS* between q and g. As a result, $\xi \geq 0$ holds.

Next, we prove that $\xi \leq |E(q)| - |E(q')|$. Consider the graph q', whose edge set can be divided into two parts: $E(\mathsf{mcs}(q', g))$ and $E(q') - E(\mathsf{mcs}(q', g))$. We check whether the edges in q' occur in $E(\mathsf{mcs}(q, g))$, and further divide the part $E(\mathsf{mcs}(q', g))$ into $E(\mathsf{mcs}(q', g)) \cap E(\mathsf{mcs}(q, g))$ and $E(\mathsf{mcs}(q', g)) \setminus E(\mathsf{mcs}(q, g))$, denoted as S_0 and S_1, respectively. Similarly, we divide the part $E(q') - E(\mathsf{mcs}(q', g))$ into $(E(q') - E(\mathsf{mcs}(q', g))) \cap E(\mathsf{mcs}(q, g))$ and $(E(q') - E(\mathsf{mcs}(q', g))) \setminus E(\mathsf{mcs}(q, g))$, denoted as T_0 and T_1, respectively. The relationship for the above subsets is illustrated in Fig. 3.

We claim that $|S_0| + |T_0| \leq |E(\mathsf{mcs}(q', g)|$. Indeed, suppose that $|S_0| + |T_0| > |E(\mathsf{mcs}(q', g)|$, from the definition of S_0 and T_0, we can derive that $S_0 \subseteq E(\mathsf{mcs}(q, g))$ and $T_0 \subseteq E(\mathsf{mcs}(q, g))$. It follows that $S_0 \cup T_0 \subseteq E(\mathsf{mcs}(q, g))$. Therefore, a common subgraph q^* between q' and g can be induced by the set $S_0 \cup T_0$, and $E(q^*) = |S_0| + |T_0| > |E(\mathsf{mcs}(q', g))|$, which contradicts the fact that $\mathsf{mcs}(q', g)$ is the maximum common subgraph between q' and g. Thus the claim is established. Moreover, we have:

$$\begin{aligned} |S_1| + |T_1| =\ & E(\mathsf{mcs}(q', g)) \setminus E(\mathsf{mcs}(q, g)) \\ & + (E(q') - E(\mathsf{mcs}(q', g))) \setminus E(\mathsf{mcs}(q, g)) \\ =\ & E(q') \setminus E(\mathsf{mcs}(q, g)) \\ =\ & E(q') \cap (E(q) - E(\mathsf{mcs}(q, g)) \\ \leq\ & |E(q)| - |E(\mathsf{mcs}(q, g)| \end{aligned}$$

Thus, we have

$$\begin{aligned} |E(q')| =\ & |S_0| + |S_1| + |T_0| + |T_1| \\ =\ & (|S_0| + |T_0|) + (|S_1| + |T_1|) \\ \leq\ & |E(\mathsf{mcs}(q', g)| + |E(q)| - |E(\mathsf{mcs}(q, g)| \end{aligned}$$

It follows that $\xi = |E(\mathsf{mcs}(q, g)| - |E(\mathsf{mcs}(q', g)| \leq |E(q)| - |E(q')|$. This completes the proof. □

Theorem 4.1: *Given two graphs q and g with dissimilarity $\delta_1(q, g) = \alpha$, for any subgraph $q' \subseteq q$, the following equation holds:*

$$\alpha - \varepsilon_{1l} \leq \delta_1(q', g) \leq \alpha + \varepsilon_{1r}$$

where $\varepsilon_{1l} = \frac{|E(q)| - \min\{|E(q')|, |E(g)|\}}{\min\{|E(q')|, |E(g)|\}}(1-\alpha)$, $\varepsilon_{1r} = \frac{|E(q)| - |E(q')|}{|E(g)|}$.

Proof: We prove this in all three cases regarding the relationship among the sizes of g, q, and q'.
(Case-1): $|E(g)| \geq |E(q)|$. Since $q' \subseteq q$, we have $|E(q')| \leq |E(q)| \leq |E(g)|$. Hence, we have

$$\begin{aligned}\delta_1(q', g) &= 1 - \frac{|E(\mathsf{mcs}(q', g))|}{\max\{|E(q')|, |E(g)|\}} \\ &= 1 - \frac{|E(\mathsf{mcs}(q, g))| - \xi}{|E(g)|} = \alpha + \frac{\xi}{|E(g)|}\end{aligned}$$

With Lemma 4.1, we have $\alpha \leq \delta_1(q', g) \leq \alpha + \frac{|E(q)|-|E(q')|}{|E(g)|}$
(Case-2): $|E(q')| \geq |E(g)|$. Since $q' \subseteq q$, we have $|E(q)| \geq |E(q')| \geq |E(g)|$. Moreover, we have $|E(\mathsf{mcs}(q, g))| = (1 - \alpha)|E(q)|$ by Eq. (1). It follows that

$$\begin{aligned}\delta_1(q', g) &= 1 - \frac{|E(\mathsf{mcs}(q', g))|}{|E(q')|} = 1 - \frac{|E(\mathsf{mcs}(q, g))| - \xi}{|E(q')|} \\ &= 1 - \frac{(1-\alpha)|E(q)|}{|E(q')|} + \frac{\xi}{|E(q')|} \\ &= \alpha - \frac{|E(q)| - |E(q')|}{|E(q')|}(1-\alpha) + \frac{\xi}{|E(q')|}\end{aligned}$$

By Lemma 4.1, we have

$$\alpha - \frac{|E(q)| - |E(q')|}{|E(q')|}(1-\alpha) \leq \delta_1(q', g) \leq \alpha + \frac{|E(q)| - |E(q')|}{|E(q')|}\alpha$$

(Case-3): $|E(q)| > |E(g)| > |E(q')|$. Similar to the proof for Case-2, we have

$$\delta_1(q', g) = \alpha - \frac{|E(q)| - |E(g)|}{|E(g)|}(1-\alpha) + \frac{\xi}{|E(g)|}$$

By Lemma 4.1, we have

$$\alpha - \frac{|E(q)| - |E(g)|}{|E(g)|}(1-\alpha) \leq \delta_1(q', g) \leq \alpha + \theta$$

where $\theta = \frac{|E(q)|-|E(g)|}{|E(g)|}\alpha + \frac{|E(g)|-|E(q')|}{|E(g)|}$.
Then, consider the overall lower bound and upper bound of $\delta_1(q', g)$. For the lower bound, clearly, the lower bounds in Case-2 and Case-3 are smaller than α in Case-1. Comparing the lower bounds in Case-2 and Case-3, their difference is as follows.

$$\begin{aligned}&\alpha - \frac{|E(q)| - |E(q')|}{|E(q')|}(1-\alpha) - (\alpha - \frac{|E(q)| - |E(g)|}{|E(g)|}(1-\alpha)) \\ &= (\frac{|E(q)| - |E(g)|}{|E(g)|} - \frac{|E(q)| - |E(q')|}{|E(q')|})(1-\alpha) \\ &= \frac{|E(q)|(|E(q')| - |E(g)|)}{|E(g)||E(q')|}(1-\alpha)\end{aligned}$$

Thus, we have $\varepsilon_{1l} = \frac{|E(q)| - \min\{|E(q')|, |E(g)|\}}{\min\{|E(q')|, |E(g)|\}}(1-\alpha)$

For the upper bound, since the upper bounds have the same first term in the above three cases, we only need to compare their second terms. For the second term in Case-2, we have

$$\frac{|E(q)| - |E(q')|}{|E(q')|}\alpha \leq \frac{|E(q)| - |E(q')|}{|E(g)|}\alpha \leq \frac{|E(q)| - |E(q')|}{|E(g)|}$$

This implies that the upper bound in Case-2 is smaller than that in case-1. Similarly, for θ in Case-3, we have

$$\begin{aligned}\theta &= \frac{|E(q)|-|E(g)|}{|E(g)|}\alpha + \frac{|E(g)| - |E(q)| + |E(q)| - |E(q')|}{|E(g)|} \\ &= \frac{|E(q)| - |E(g)|}{|E(g)|}(\alpha - 1) + \frac{|E(q)| - |E(q')|}{|E(g)|} \leq \frac{|E(q)| - |E(q')|}{|E(g)|}\end{aligned}$$

This shows that the upper bound in Case-3 is smaller than that in Case-1. Thus we have $\varepsilon_{1r} = \frac{|E(q)|-|E(q')|}{|E(g)|}$. Therefore, we conclude that $\alpha - \varepsilon_{1l} \leq \delta_1(q', g) \leq \alpha + \varepsilon_{1r}$ for any $q' \subseteq q$. □

Theorem 4.2: *Given two graphs q and g with dissimilarity $\delta_2(q, g) = \alpha$, for any subgraph $q' \subseteq q$, the following equation holds:*

$$\alpha - (1-\alpha)\varepsilon_2 \leq \delta_2(q', g) \leq \alpha + (1+\alpha)\varepsilon_2$$

where $\varepsilon_2 = \frac{|E(q)|-|E(q')|}{|E(q')|+|E(g)|}$.
Proof: Since $\delta_2(q, g) = \alpha$, we have $2|E(\mathsf{mcs}(q, g))| = (1 - \alpha)(|E(q)| + |E(g)|)$ by Eq. (2). Thus, we have

$$\begin{aligned}\delta_2(q', g) &= 1 - \frac{2|E(\mathsf{mcs}(q', g))|}{|E(q')| + |E(g)|} \\ &= 1 - \frac{2(|E(\mathsf{mcs}(q, g))| - \xi)}{|E(q')| + |E(g)|} \\ &= 1 - \frac{(1-\alpha)(|E(q)| + |E(g)|)}{|E(q')| + |E(g)|} + \frac{2\xi}{|E(q')| + |E(g)|} \\ &= \alpha - \frac{(1-\alpha)(|E(q)| - |E(q')|)}{|E(q')| + |E(g)|} + \frac{2\xi}{|E(q')| + |E(g)|}\end{aligned}$$

By Lemma 4.1, we have $\alpha - (1-\alpha)\varepsilon_2 \leq \delta_2(q', g) \leq \alpha + (1+\alpha)\varepsilon_2$. This completes the proof. □

Note that when q' is very close to q, ε_{1l}, ε_{1r}, and ε_2 are very small. Next, we discuss the bound when they are mapped onto the corresponding multidimensional space, $\mathcal{F}$. For three graphs q, g, and $q' \subseteq q$, we use $\mathcal{F}(q)$, $\mathcal{F}(g)$, and $\mathcal{F}(q')$ to denote the sets of features contained by them respectively. The mapped binary vectors can be denoted by $\mathbf{y}_q$, $\mathbf{y}_g$, and $\mathbf{y}_{q'}$ correspondingly. We have the following theorem.

Theorem 4.3: *Given two graphs q and g, if their mapped distance is $d(\mathbf{y}_q, \mathbf{y}_g) = \beta$ in the multidimensional space $\mathcal{F}$, then for any subgraph $q' \subseteq q$, the following equation holds:*

$$\beta - \sqrt{t/p} \leq d(\mathbf{y}_{q'}, \mathbf{y}_g) \leq \beta + \sqrt{t/p}$$

where $t = |\mathcal{F}(q)| - |\mathcal{F}(q')|$ and $p = |\mathcal{F}|$.
Proof: Since $\mathbf{y}_q$ and $\mathbf{y}_g$ are binary vectors, the mapped distance between q and g can be rewritten as

$$\begin{aligned}d(\mathbf{y}_q, \mathbf{y}_g) &= \frac{1}{\sqrt{p}}\big(\Sigma_{r=1}^{|\mathcal{F}|}(y_{qr} - y_{gr})^2\big)^{\frac{1}{2}} \\ &= \frac{1}{\sqrt{p}}\big(|\mathcal{F}(q)| + |\mathcal{F}(g)| - 2|\mathcal{F}(q) \cap \mathcal{F}(g)|\big)^{\frac{1}{2}} = \beta\end{aligned}$$

Then we have $|\mathcal{F}(q)|+|\mathcal{F}(g)|-2|\mathcal{F}(q)\cap\mathcal{F}(g)| = \beta^2 p$. Similarly, we have $d(\mathbf{y}_{q'}, \mathbf{y}_g) = \frac{1}{\sqrt{p}}\big(|\mathcal{F}(q')| + |\mathcal{F}(g)| - 2|\mathcal{F}(q') \cap F(g)|\big)^{\frac{1}{2}}$ for q'. Let $t = |\mathcal{F}(q)| - |\mathcal{F}(q')|$. Since $q' \subseteq q$, we have $\mathcal{F}(q') \subseteq \mathcal{F}(q)$, which means that each feature contained by q' is also contained by q. Then we have $t = |\mathcal{F}(q) - \mathcal{F}(q')| = |\mathcal{F}(q)| - |\mathcal{F}(q')|$. Thus we can rewrite $d(\mathbf{y}_{q'}, \mathbf{y}_g)$ as

$$\begin{aligned}d(\mathbf{y}_{q'}, \mathbf{y}_g) &= \frac{1}{\sqrt{p}}\big(|\mathcal{F}(q')| + |\mathcal{F}(g)| \\ &\quad - 2|(\mathcal{F}(q) - (\mathcal{F}(q) - \mathcal{F}(q'))) \cap \mathcal{F}(g)|\big)^{\frac{1}{2}} \\ &= \frac{1}{\sqrt{p}}\big(|\mathcal{F}(q)| - t + |\mathcal{F}(g)| - 2|\mathcal{F}(q) \cap \mathcal{F}(g)| \\ &\quad + 2|(\mathcal{F}(q) - \mathcal{F}(q')) \cap \mathcal{F}(g)|\big)^{\frac{1}{2}} \\ &= \frac{1}{\sqrt{p}}\big(\beta^2 p - t + 2|(\mathcal{F}(q) - \mathcal{F}(q')) \cap \mathcal{F}(g)|\big)^{\frac{1}{2}} \\ &= \big(\beta^2 - \frac{t}{p} + \frac{2}{p}|(\mathcal{F}(q) - \mathcal{F}(q')) \cap \mathcal{F}(g)|\big)^{\frac{1}{2}}\end{aligned}$$

In fact, term $|(\mathcal{F}(q) - \mathcal{F}(q')) \cap \mathcal{F}(g)|$ is determined by the number of bits of value 1 in the corresponding t bits of $\mathbf{y}_g$. We obtain its bounds by considering two extreme cases. One is that the values in these t bits of $\mathbf{y}_g$ are all 1. We have $|(\mathcal{F}(q) - \mathcal{F}(q')) \cap \mathcal{F}(g)| = t$. The other is that the values in these t bits of $\mathbf{y}_g$ are all 0, which leads to $|(\mathcal{F}(q) - \mathcal{F}(q')) \cap \mathcal{F}(g)| = 0$. We have $0 \leq |(\mathcal{F}(q) -$

$\mathcal{F}(q')) \cap \mathcal{F}(g)| \leq t$. It follows that $(\beta^2 - t/p)^{\frac{1}{2}} \leq d(\mathbf{y}_{q'}, \mathbf{y}_g) \leq (\beta^2 + t/p)^{\frac{1}{2}}$. Since $(\beta^2 - t/p)^{\frac{1}{2}} \geq \beta - \sqrt{t/p}$ and $(\beta^2 + t/p)^{\frac{1}{2}} \leq \beta + \sqrt{t/p}$, we have $\beta - \sqrt{t/p} \leq d(\mathbf{y}_{q'}, \mathbf{y}_g) \leq \beta + \sqrt{t/p}$. □

Corollary 4.1: *Given two graphs q and g, if $\delta(q, g) = \alpha$ and $d(\mathbf{y}_q, \mathbf{y}_g) = \beta$ and $\lambda = \alpha/\beta$, then for any graph $q' \subseteq q$, we have*

$$\lambda_1 = \frac{\delta_1(q', g)}{d(\mathbf{y}_{q'}, \mathbf{y}_g)} \in [\frac{\alpha - \varepsilon_{1l}}{(\beta + \sqrt{t/p})}, \frac{\alpha + \varepsilon_{1r}}{(\beta - \sqrt{t/p})}]$$

$$\lambda_2 = \frac{\delta_2(q', g)}{d(\mathbf{y}_{q'}, \mathbf{y}_g)} \in [\frac{\alpha - (1-\alpha)\varepsilon_2}{(\beta + \sqrt{t/p})}, \frac{\alpha + (1+\alpha)\varepsilon_2}{(\beta - \sqrt{t/p})}]$$

This corollary is obtained based on Theorems 4.1, 4.2, and 4.3. From this corollary, we can see that if the approximate ratio is good for q and g, then it can also be well bounded for a subgraph $q' \subseteq q$ and g . Similarly, we derive another corollary as follows.

Corollary 4.2: *Given two graphs q' and g, if $\delta(q', g) = \alpha'$ and $d(\mathbf{y}'_q, \mathbf{y}_g) = \beta'$ and $\lambda' = \alpha'/\beta'$, then for any graph $q \supseteq q'$, we have*

$$\lambda'_1 = \frac{\delta_1(q, g)}{d(\mathbf{y}_q, \mathbf{y}_g)} \in [\frac{\alpha' - \varepsilon_{1r}}{(\beta' + \sqrt{t/p})}, \frac{\alpha' + \varepsilon_{1l}}{(\beta' - \sqrt{t/p})}]$$

$$\lambda'_2 = \frac{\delta_2(q, g)}{d(\mathbf{y}_q, \mathbf{y}_g)} \in [\frac{\alpha' - \varepsilon_2}{(\beta' + \sqrt{t/p})(1+\varepsilon_2)}, \frac{\alpha' + \varepsilon_2}{(\beta' - \sqrt{t/p})(1+\varepsilon_2)}]$$

Proof: According to Theorem 4.1, the following formula holds:

$$\alpha' - \varepsilon_{1l} \leq \delta_1(q, g) \leq \alpha' + \varepsilon_{1r}$$

According to Theorem 4.2, we can derive the following formula:

$$\frac{\alpha' - \varepsilon_2}{1 + \varepsilon_2} \leq \delta_2(q, g) \leq \frac{\alpha' + \varepsilon_2}{1 + \varepsilon_2}$$

According to Theorem 4.3 , we have:

$$\beta' - \sqrt{t/p} \leq d(\mathbf{y}_q, \mathbf{y}_g) \leq \beta' + \sqrt{t/p}$$

Corollary 4.2 can be obtained directly. □

Discussion: In the following, we show that the selection of the high quality dimensions $\mathcal{F}$ from the graph database $\mathcal{D}_\mathcal{G}$ which lead to the high quality of distance-preserving can also lead to the high quality of structure-preserving. Suppose that the quality of mapping for a graph $q(\in \mathcal{D}_\mathcal{G})$ is bounded, we consider a query graph q' ($\notin \mathcal{D}_\mathcal{G}$). There are two cases:

- *(Case-1: q' is similar to q)* Since the quality of mapping for a graph $q(\in \mathcal{D}_\mathcal{G})$ is bounded, the quality of mapping for the query graph q' can also be bounded by first considering the quality of $\mathsf{mcs}(q', q)$ ($\subseteq q$) by Corollary 4.1, and then using Corollary 4.2 since $q' \supseteq \mathsf{mcs}(q', q)$. In other words, the high quality of distance-preserving leads to the high quality of structure-preserving in such a case.
- *(Case-2: q' is dissimilar to q)* In such a case, $\delta(q', q)$ is large, or in other words, $\mathsf{mcs}(q', q)$ is small. Recall that a high quality *DS-preserved* mapping should avoid generating dimensions that are highly overlapped with each other, or in other words, the dimensions generated by a high quality *DS-preserved* mapping tend to be independent. Given such high quality dimensions $\mathcal{F}$, and a small $\mathsf{mcs}(q', q)$, the number of features in $\mathcal{F}$ contained in $\mathsf{mcs}(q', q)$ tends to be small, and thus the size of $\mathcal{F}(q') \cap \mathcal{F}(q)$ tends to be small. Since $d(\mathbf{y}'_q, \mathbf{y}_q) = \frac{1}{\sqrt{p}} \left(|\mathcal{F}(q')| + |\mathcal{F}(q)| - 2 \times |\mathcal{F}(q') \cap \mathcal{F}(q)|\right)^{\frac{1}{2}}$, $d(\mathbf{y}'_q, \mathbf{y}_q)$ tends to be large. In other words, the high quality of distance-preserving also leads to the high quality of structure-preserving in such a case.

As discussed above, in either of the two cases, the high quality of distance-preserving can lead to the high quality of structure-preserving, thus, in the following, we concentrate on how to compute the set of dimensions for distance-preserving mapping.

4.2 Dimension Computation

We discuss how to identify the dimensions $\mathcal{F}$ for *DS-preserved* mapping. Here, a dimension in $\mathcal{F}$ is a subgraph taken from the set of frequent subgraphs, denoted $F = \{f_1, f_2, \ldots, f_m\}$, mined from $\mathcal{D}_\mathcal{G}$ using any existing algorithm with a given τ. Every graph $g_i \in \mathcal{D}_\mathcal{G}$ is represented by a binary feature vector $\mathbf{y}_i = (y_{i1}, y_{i2}, \ldots, y_{im})$, where $y_{ir} = 1$ if f_r is a subgraph of g_i, otherwise $y_{ir} = 0$. We need to evaluate all the features in F and choose the best $\mathcal{F}$ with p ($\leq m$) features so that the graph dissimilarity is *DS-preserved*. Note that here we use the binary vector $\mathbf{y}_i$ to model the feature space. Another possible way is to use the actual number of occurrences of a subgraph f_r as y_{ir}. However, in such a way, the number of occurrences for a certain subgraph can be exponential, which makes the vector $\mathbf{y}_i$ very unbalanced. Therefore, in this paper, we only consider $\mathbf{y}_i$ as a binary vector.

We assign F a weight vector $\mathbf{c} = (c_1, c_2, \ldots, c_m)$ to be estimated, where $c_r > 0$ ($1 \leq r \leq m$) if f_r is selected and otherwise $c_r = 0$. Then a weighted feature vector is formed for every graph $g_i \in \mathcal{D}_\mathcal{G}$, denoted as $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$, where

$$x_{ir} = y_{ir} c_r \tag{3}$$

To select the p feature dimensions, we require the weight vector $\mathbf{c}$ to satisfy the constraint $\Sigma_{r=1}^m sgn(c_r) = p$ where

$$sgn(c_r) = \begin{cases} 1 & \text{if } c_r > 0 \\ 0 & \text{if } c_r = 0 \end{cases}$$

The set of features selected to form $\mathcal{F}$ ($\subseteq F$) is given as $\mathcal{F} = \{f_r | sgn(c_r) = 1\}$, where $|\mathcal{F}| = p$.

Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be the data objects in F for the corresponding graphs g_i and g_j in $\mathcal{D}_\mathcal{G}$. The Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ is computed as

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left(\Sigma_{r=1}^m (x_{ir} - x_{jr})^2\right)^{\frac{1}{2}} = \left(\Sigma_{r=1}^m (y_{ir} c_r - y_{jr} c_r)^2\right)^{\frac{1}{2}}$$

where $\Sigma_{r=1}^m c_r^2 = 1$ so that the distance is normalized to $[0, 1]$. In the following, for simplicity, we do not consider such a constraint, since after computing $c_1, c_2, \ldots, c_m$, we can normalize them to $[0, 1]$ easily in a post-processing stage. The error between the Euclidean distance and the dissimilarity is defined as $e_{ij} = (d(\mathbf{x}_i, \mathbf{x}_j) - \delta_{ij})^2$. The total error for *DS-preserved* mapping is to sum up the errors over all g_i and g_j in $\mathcal{D}_\mathcal{G}$ as follows.

$$\mathcal{E}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \Sigma_{1 \leq i,j \leq n} (d(\mathbf{x}_i, \mathbf{x}_j) - \delta_{ij})^2 \tag{4}$$

Then, the problem of idenitying $\mathcal{F}$ of p-dimensions becomes a least square optimization problem with a constraint as follows.

$$\begin{array}{ll} \text{minimize} & \mathcal{E}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \\ \text{subject to} & \mathbf{x}_i = (y_{i1} c_1, y_{i2} c_2, \ldots, y_{im} c_m) \quad \text{for } i = 1, \ldots, n \\ & \Sigma_{r=1}^m sgn(c_r) = p \end{array} \tag{5}$$

5. THE ALGORITHMS

In this section, we study how to minimize $\mathcal{E}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ to compute $\mathcal{F}$. We propose an iterative algorithm, and discuss several optimization techniques to reduce the computational cost. We further derive an approximate algorithm to handle a large graph database.

Our algorithm, denoted as DSPM, is inspired by the majorization strategy in [36]. Unlike many traditional minimization methods, the majorization strategy iteratively generates a converging sequence of function values without a stepsize procedure that may be computationally expensive and unreliable. For the sake of completeness, we give a brief overview of majorization strategy before describing the details of our algorithms.

The basic idea of majorization is, for a complicated function $h(\mathbf{x})$ which is hard to obtain an analytical solution, we find a simpler auxiliary function, $g(\mathbf{x}, \mathbf{z})$, whose value is no smaller than $h(\mathbf{x})$. The auxiliary function, also called the majorizing function, should equal the original function at a supporting point $\mathbf{z} = \mathbf{x}$, i.e.,

$h(\mathbf{x}) = g(\mathbf{x}, \mathbf{z})$. Suppose that the minimum value of $g(\mathbf{x}, \mathbf{z})$ over $\mathbf{x}$ is attained at configuration $\overline{\mathbf{x}}$, then we have the following chain of inequalities:

$$h(\mathbf{z}) = g(\mathbf{z}, \mathbf{z}) \geq g(\overline{\mathbf{x}}, \mathbf{z}) \geq h(\overline{\mathbf{x}})$$

Here, $\overline{\mathbf{x}}$ becomes the supporting point of the next majorizing function. We iterate this process until convergence occurs due to a lower bound of the function or due to constraints. As stated in [37], the configuration of the supporting point and the parameter can be updated for Eq. (5) in the following form.

$$\overline{x}_{ir} = \frac{1}{n}\Sigma_{k=1}^{n} b_{ik} z_{kr} \tag{6}$$

$$c_r^u = \frac{\Sigma_{1\leq i,j\leq n}(\overline{x}_{ir} - \overline{x}_{jr})(y_{ir} - y_{jr})}{\Sigma_{1\leq i,j\leq n}(y_{ir} - y_{jr})^2} \tag{7}$$

where b_{ij} is defined as

$$b_{ij} = \begin{cases} -\dfrac{\delta_{ij}}{d(\mathbf{z}_i, \mathbf{z}_j)} & \text{for } i \neq j \text{ and } d(\mathbf{z}_i, \mathbf{z}_j) \neq 0 \\ 0 & \text{for } i \neq j \text{ and } d(\mathbf{z}_i, \mathbf{z}_j) = 0 \end{cases} \tag{8}$$
$$b_{ii} = -\Sigma_{j=1, j\neq i}^{n} b_{ij}$$

for $i, j = 1, \ldots, n$, and $r = 1, \ldots, m$.

Let k be the number of iterations, and recall that $n = |\mathcal{D}_\mathcal{G}|$ and $m = |F|$. In the majorization iteration, there are three costly operations: compute $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$, update the configuration $\overline{\mathbf{x}}$, and update the weight vector $\mathbf{c}$. According to Eq. (4), Eq. (6), and Eq. (7), the time complexity of these three parts are all $O(m \cdot n^2)$. Thus the overall complexity of majorization method is $O(k{\cdot}m{\cdot}n^2)$. Therefore, a straightforward implementation of the algorithm will induce very high computational cost when n and m are large. In the following, we propose a fast algorithm DSPM by exploiting the characteristics of the graph features to optimize the computation.

5.1 The DSPM Algorithm

We first give the outline of DSPM in Algorithm 1 and will discuss the optimization techniques later. For simplicity, we use $\mathcal{E}_k$ to denote the value of the objective function $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$ in the k-th iteration. The initial objective function $\mathcal{E}_0$ is set to infinity in line 2. The weight parameter c_r is initialized as $1/\sqrt{m}$ for each feature f_r in line 3. Lines 4-6 initialize the binary vector $\mathbf{y}_i$ for each graph $g_i \in \mathcal{D}_\mathcal{G}$. We update the value of z_{ir} by Eq. (3) and compute the current value of the objective function $\mathcal{E}$ by function Computeobj($\mathcal{D}_\mathcal{G}$, F, $[z_{ir}]_{n\times m}$) in lines 7-8. Then, we compute $\overline{x}_{ir}$ by function Updatexbar ($[z_{ir}]_{n\times m}$) in line 10, and update $\mathbf{c}$ by function Updatec ($\mathcal{D}_\mathcal{G}, F, [\overline{x}_{ir}]_{n\times m}$) in line 11. A new configuration of z_{ir} is obtained in line 12. Such process repeats until the difference between the objective function values in two consecutive iterations is smaller than a threshold ϵ or the maximum number of iterations is reached.

Next we show how to derive a simpler formula for the weight vector $\mathbf{c}$ by considering the characteristics of subgraph features, and present the optimization techniques used in the three functions Computeobj, Updatexbar, and Updatec in DSPM. Note that these optimization functions will not affect the convergence property of our algorithm DSPM.

5.1.1 Simplify the Formula of $\mathbf{c}$

We discuss how to derive a simpler formula of $\mathbf{c}$ by exploring the characteristics of subgraph features and the binary vectors for each graph. We first show a proposition as follows.

Proposition 5.1: *Given a graph database $\mathcal{D}_\mathcal{G}$, a frequent subgraph set F, and the binary vector $\mathbf{y}_i$ for each graph $g_i \in \mathcal{D}_\mathcal{G}$, for each $f_r \in F$, we have $\sum_{i=1}^{n} y_{ir} = |sup(f_r)|$ and $\sum_{i=1}^{n} y_{ir}^2 = |sup(f_r)|$.*

Algorithm 1 DSPM

Input: A graph database $\mathcal{D}_\mathcal{G}$ and a frequent feature set F.
Output: A selected feature set $\mathcal{F}$.
1: $k \leftarrow 1$;
2: $\mathcal{E}_0 \leftarrow +\infty$;
3: $c_r \leftarrow 1/\sqrt{m}$ for $r = 1, \ldots, m$;
4: $y_{ir} \leftarrow 0$ for $i = 1, \ldots, n, r = 1, \ldots, m$;
5: **for all** $f_r \in F$ and $g_i \in sup(f_r)$ **do**
6: $\quad y_{ir} \leftarrow 1$;
7: $z_{ir} \leftarrow y_{ir}c_r$ for $i = 1, \ldots, n, r = 1, \ldots, m$;
8: $\mathcal{E}_k \leftarrow$ Computeobj($\mathcal{D}_\mathcal{G}$, F, $[z_{ir}]_{n\times m}$) ;
9: **while** $\mathcal{E}_{k-1} - \mathcal{E}_k > \epsilon$ **and** $k \leq$ maximum iteration number **do**
10: $\quad [\overline{x}_{ir}]_{n\times m} \leftarrow$ Updatexbar($[z_{ir}]_{n\times m}$) ;
11: $\quad \mathbf{c} \leftarrow$ Updatec($\mathcal{D}_\mathcal{G}$, F, $[\overline{x}_{ir}]_{n\times m}$) ;
12: $\quad z_{ir} \leftarrow y_{ir}c_r$ for $i = 1, \ldots, n, r = 1, \ldots, m$;
13: $\quad k \leftarrow k + 1$;
14: $\quad \mathcal{E}_k \leftarrow$ Computeobj($\mathcal{D}_\mathcal{G}$, F, $[z_{ir}]_{n\times m}$) ;
15: $\mathcal{F} \leftarrow p$ features with largest c_r;
16: **return** $\mathcal{F}$;

Algorithm 2 Updatec ($\mathcal{D}_\mathcal{G}, F, [\overline{x}_{ir}]_{n\times m}$)

1: **for all** $f_r \in F$ **do**
2: $\quad c_r \leftarrow 0$;
3: $\quad$ **for all** $g_i \in \mathcal{D}_\mathcal{G}$ **do**
4: $\quad\quad$ **if** $g_i \in \mathcal{I}_r^F$ **then**
5: $\quad\quad\quad c_r \leftarrow c_r + \frac{\overline{x}_{ir}(1-|sup(f_r)|)}{|sup(f_r)|(n-|sup(f_r)|)}$;
6: $\quad\quad$ **else**
7: $\quad\quad\quad c_r \leftarrow c_r + \frac{\overline{x}_{ir}(0-|sup(f_r)|)}{|sup(f_r)|(n-|sup(f_r)|)}$;
8: **return** $\mathbf{c}$;

Recall that $sup(f) = \{g_i \mid f \subseteq g_i, g_i \in \mathcal{D}_\mathcal{G}\}$. The proof of this proposition is omitted as it can be easily derived from $sup(f_r)$ and the characteristics of the binary vector $\mathbf{y}_i$ for each $g_i \in \mathcal{D}_\mathcal{G}$. Based on this proposition, we have the following theorem.

Theorem 5.1: *The calculation of the weight vector $\mathbf{c}$ in Eq. (7) can be simplified as*

$$c_r^u = \frac{\sum_{i=1}^{n} \overline{x}_{ir}(ny_{ir} - |sup(f_r)|)}{|sup(f_r)|(n - |sup(f_r)|)} \tag{9}$$

Proof: We prove this by considering the numerator and denominator on the right side of Eq. (7) respectively. First, we consider the numerator which can be rewritten as

$$\begin{aligned} & \Sigma_{1\leq i,j\leq n}(\overline{x}_{ir} - \overline{x}_{jr})(y_{ir} - y_{jr}) \\ =\ & \Sigma_{1\leq i,j\leq n}(\overline{x}_{ir}y_{ir} + \overline{x}_{jr}y_{jr} - \overline{x}_{jr}y_{ir} - \overline{x}_{ir}y_{jr}) \\ =\ & 2n\Sigma_{i=1}^{n}\overline{x}_{ir}y_{ir} - 2\Sigma_{i=1}^{n}\overline{x}_{ir}\Sigma_{j=1}^{n}y_{jr} \\ =\ & 2\Sigma_{i=1}^{n}\overline{x}_{ir}(ny_{ir} - |sup(f_r)|) \text{ (by Proposition 5.1)} \end{aligned}$$

Then we rewrite denominator as

$$\begin{aligned} & \Sigma_{1\leq i,j\leq n}(y_{ir} - y_{jr})^2 \\ =\ & \Sigma_{1\leq i,j\leq n}(y_{ir}^2 + y_{jr}^2 - 2y_{ir}y_{jr}) \\ =\ & 2n\Sigma_{i=1}^{n}y_{ir}^2 - 2\Sigma_{i=1}^{n}y_{ir}\Sigma_{j=1}^{n}y_{jr} \\ =\ & 2n|sup(f_r)| - 2\Sigma_{i=1}^{n}y_{ir}|sup(f_r)| \text{ (by Proposition 5.1)} \\ =\ & 2|sup(f_r)|(n - |sup(f_r)|) \text{ (by Proposition 5.1)} \end{aligned}$$

Thus we have

$$c_r^u = \frac{\sum_{i=1}^{n} \overline{x}_{ir}(ny_{ir} - |sup(f_r)|)}{|sup(f_r)|(n - |sup(f_r)|)}$$

This completes the proof. □

5.1.2 Optimize the Computation of the Functions

We introduce two data structures used in the algorithms before presenting the techniques to optimize the computation of the three functions Updatec, Updatexbar, and Computeobj used in DSPM. For each feature $f_r \in F$, we construct an inverted list $\mathcal{I}_r^F$ to store the set of graphs which contain the feature f_r, i.e., $\mathcal{I}_r^F = \{g_i | g_i \supseteq f_r, g_i \in \mathcal{D}_\mathcal{G}\}$. For each graph g_i in the graph database $\mathcal{D}_\mathcal{G}$, we also construct an inverted list $\mathcal{I}_i^\mathcal{G}$ to store the set of frequent features that is contained in graph g_i, i.e., $\mathcal{I}_i^\mathcal{G} = \{f_r | f_r \subseteq g_i, f_r \in F\}$.

Algorithm 3 Updatexbar $([z_{ir}]_{n\times m})$

1: **for all** $g_i, g_j \in \mathcal{D}_\mathcal{G}$ **do**
2: compute b_{ij} by Eq. (8);
3: **for all** $g_i \in \mathcal{D}_\mathcal{G}$ **do**
4: **for all** $f_r \in F$ **do**
5: $\overline{x}_{ir} \leftarrow 0$;
6: **for all** $g_k \in \mathcal{I}_r^F$ **do**
7: $\overline{x}_{ir} \leftarrow \overline{x}_{ir} + \frac{1}{n} b_{ik} z_{kr}$;
8: **return** $[\overline{x}_{ir}]_{n\times m}$;

Algorithm 4 Computeobj $(\mathcal{D}_\mathcal{G}, F, [z_{ir}]_{n\times m})$

1: $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n) \leftarrow 0$
2: **for all** $g_i, g_j \in \mathcal{D}_\mathcal{G}$ **do**
3: $d(\mathbf{z}_i, \mathbf{z}_j) \leftarrow 0$;
4: **for all** $f_r \in \mathcal{I}_i^\mathcal{G} \bigcup \mathcal{I}_j^\mathcal{G} - \mathcal{I}_i^\mathcal{G} \bigcap \mathcal{I}_j^\mathcal{G}$ **do**
5: $d(\mathbf{z}_i, \mathbf{z}_j) \leftarrow d(\mathbf{z}_i, \mathbf{z}_j) + c_r^2$;
6: $d(\mathbf{z}_i, \mathbf{z}_j) \leftarrow d(\mathbf{z}_i, \mathbf{z}_j)^{\frac{1}{2}}$;
7: $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n) \leftarrow \mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n) + (d(\mathbf{z}_i, \mathbf{z}_j) - \delta_{ij})^2$
8: **return** $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$;

Function Updatec. Algorithm 2 shows how to compute Updatec $(\mathcal{D}_\mathcal{G}, F, [\overline{x}_{ir}]_{n\times m})$ to simplify the update of $\mathbf{c}$. According to Theorem 5.1, c_r can be calculated by considering two cases of y_{ir}: $y_{ir} = 0$ and $y_{ir} = 1$. At the beginning, we initially set c_r to 0 for each feature $f_r \in F$ in line 2. Then for each graph g_i, we check if it is in the inverted list $\mathcal{I}_r^F$. If yes, the value of y_{ir} is 1, and c_r is increased by $\frac{\overline{x}_{ir}(1-|sup(f_r)|)}{|sup(f_r)|(n-|sup(f_r)|)}$; otherwise, $y_{ir} = 0$, and c_r is increased by $\frac{\overline{x}_{ir}(0-|sup(f_r)|)}{|sup(f_r)|(n-|sup(f_r)|)}$.

Function Updatexbar. The function Updatexbar $([z_{ir}]_{n\times m})$ to compute the value of $\overline{x}_{ir}$ is shown in Algorithm 3. First, we compute b_{ij} by Eq. (8) for any two graphs $g_i, g_j \in \mathcal{D}_\mathcal{G}$ in lines 1-2. Then we start to compute $\overline{x}_{ir}$ for each graph $g_i \in \mathcal{D}_\mathcal{G}$ and $f_r \in F$ based on b_{ij} in lines 3-7. As shown in Eq. (6), in each computation we need to add the value of $b_{ik}z_{kr}$ for each graph $g_k \in \mathcal{D}_\mathcal{G}$. Note that z_{kr} is 0 if feature f_r is not contained in graph g_k. This means that the computation of $\overline{x}_{ir}$ will not be affected if we skip adding $b_{ik}z_{kr}$ for those $g_k \notin \mathcal{I}_r^F$. Therefore, we only consider the graphs $g_k \in \mathcal{I}_r^F$ for the computation of $\overline{x}_{ir}$ in lines 6-7.

Function Computeobj. We give the algorithm Computeobj $(\mathcal{D}_\mathcal{G}$, F, $[z_{ir}]_{n\times m})$ to compute the value of the objective function $\mathcal{E}_k$ in each iteration. According to Eq. (4), we need to calculate $d(\mathbf{z}_i, \mathbf{z}_j)$ for each pair of graphs $g_i, g_j \in \mathcal{D}_\mathcal{G}$ by $\left(\Sigma_{r=1}^m (y_{ir}c_r - y_{jr}c_r)^2\right)^{\frac{1}{2}}$. In fact, we only need to consider the feature f_r with $y_{ir} \neq y_{jr}$ to calculate $d(\mathbf{z}_i, \mathbf{z}_j)$ as the distance will not change if $y_{ir} = y_{jr}$. The algorithm is shown in Algorithm 4. Initially, $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$ is set to be 0 in line 1. Then, for any two graphs $g_i, g_j \in \mathcal{D}_\mathcal{G}$, we do not examine all the features to compute their distance. Instead, we only examine their inverted lists $\mathcal{I}_i^\mathcal{G}$ and $\mathcal{I}_j^\mathcal{G}$, and increase the distance for the feature which is contained in only one of these two lists in lines 4-5. Finally, we increase the value of $\mathcal{E}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$ by $(d(\mathbf{z}_i, \mathbf{z}_j) - \delta_{ij})^2$ in line 7 according to Eq. (4).

Theorem 5.2: *The time complexity of Algorithm 1 is* $O(k \cdot n \cdot (m \cdot |\mathcal{I}^F|_{max} + n \cdot |\mathcal{I}^\mathcal{G}|_{max}))$, *where* $|\mathcal{I}^F|_{max} = \max_{1\leq r\leq m} |\mathcal{I}_r^F|$, *and* $|\mathcal{I}^\mathcal{G}|_{max} = \max_{1\leq i,j\leq n} |\mathcal{I}_i^\mathcal{G} \bigcup \mathcal{I}_j^\mathcal{G}|$.

Proof: The time complexity of Updatec $(\mathcal{D}_\mathcal{G}$, F, $[\overline{x}_{ir}]_{n\times m})$ in Algorithm 2 is $O(m \cdot n)$. The time complexity of Updatexbar $([z_{ir}]_{n\times m})$ in Algorithm 3 is $O(n^2 + m \cdot n \cdot |\mathcal{I}^F|_{max})$. Function Computeobj $(\mathcal{D}_\mathcal{G}$, F, $[\overline{z}_{ij}]_{n\times m})$ in Algorithm 4 needs $O(n^2 \cdot |\mathcal{I}^\mathcal{G}|_{max})$ time. Thus, the overall complexity of DSPM in Algorithm 1 is $O(k \cdot (m \cdot n + n^2 + m \cdot n \cdot |\mathcal{I}^F|_{max} + n^2 \cdot |\mathcal{I}^\mathcal{G}|_{max})) = O(k \cdot n \cdot (m \cdot |\mathcal{I}^F|_{max} + n \cdot |\mathcal{I}^\mathcal{G}|_{max}))$. □

Algorithm 5 DSPMap

Input: A graph database $\mathcal{D}_\mathcal{G}$ and a frequent feature set F.
Output: A weight vector $\mathbf{c}$.
1: $\mathcal{P} \leftarrow$Partition $(\mathcal{D}_\mathcal{G}, F)$;
2: $\mathbf{c} \leftarrow$ Computec $(\mathcal{P}, F)$;
3: **return** $\mathbf{c}$;

Algorithm 6 Computec $(\mathcal{P}, F)$

1: **if** $|\mathcal{P}| = 1$ **then**
2: $\mathcal{D}'_\mathcal{G} \leftarrow P$;
3: $F' \leftarrow \{f_r | f_r \in F, sup(f_r) \cap \mathcal{D}'_\mathcal{G} \neq \emptyset\}$;
4: $\mathbf{c} \leftarrow$ DSPM$(\mathcal{D}'_\mathcal{G}, F')$;
5: **else**
6: $\mathbf{c}_l \leftarrow$ Computec $(\mathcal{P}_l, F)$;
7: $\mathbf{c}_r \leftarrow$ Computec $(\mathcal{P}_r, F)$;
8: $\mathbf{c}_o \leftarrow$ DSPM $(\mathcal{D}_{\mathcal{G}_o}, F'_o)$;
9: $\mathbf{c} \leftarrow \mathbf{c}_l + \mathbf{c}_r + \mathbf{c}_o$;
10: **return** $\mathbf{c}$;

Discussion: In practice, the computation cost of Algorithm 1 will be much smaller than the worst time complexity shown in Theorem 5.2. Although the worst time complexity is bounded by $|\mathcal{I}^F|_{max}$ and $|\mathcal{I}^\mathcal{G}|_{max}$, which might be as large as n and m in the worst case, the computation time in practice is actually determined by the average sizes of the indexes, $|\mathcal{I}^F|_{avg}$ and $|\mathcal{I}^\mathcal{G}|_{avg}$, which are far smaller than n and m respectively, i.e., $|\mathcal{I}^F|_{avg} \ll n$ and $|\mathcal{I}^\mathcal{G}|_{avg} \ll m$. Thus the practical computation time can be reduced significantly.

5.2 The Approximate Algorithm DSPMap

In the previous subsection, we present the algorithm DSPM with several optimization techniques to reduce the computational cost. However, DSPM consumes a lot of memory if we deal with a very large graph database. It's because we need to store the configuration matrix $[\overline{x}_{ir}]_{n\times m}$ as well as the dissimilarity/distance for each pair of graphs, which needs $O(n \cdot (n+m))$ space in total. Thus, to make the algorithm more scalable on large graph databases, we further propose an approximate algorithm DSPMap which can reduce both space and time consumption. The computational complexity of DSPMap increases linearly with the size of the graph database.

We give the outline of DSPMap in Algorithm 5, which includes two phases. In the first phase, we partition the graph database into $n_p = \lceil \frac{n}{b} \rceil$ parts, denoted as $\mathcal{P} = \{\mathcal{D}_{\mathcal{G}1}, \mathcal{D}_{\mathcal{G}2}, \ldots, \mathcal{D}_{\mathcal{G}n_p}\}$, where b is the size of each partition. We give our partition method in Algorithm 7 which will be explained later. The second part is to compute the parameter $\mathbf{c}$. Here, we compute $\mathbf{c}$ in a recursive way. As shown in Algorithm 6, we divide the partition into the left part $\mathcal{P}_l = \{\mathcal{D}_{\mathcal{G}i} | 1 \leq i \leq \lceil \frac{n_p}{2} \rceil\}$ and the right part $\mathcal{P}_r = \mathcal{P} \setminus \mathcal{P}_l$, and recursively compute their weight vectors, $\mathbf{c}_l$ and $\mathbf{c}_r$. We then randomly select two parts, $\mathcal{D}_{\mathcal{G}_l}$ and $\mathcal{D}_{\mathcal{G}_r}$, from $\mathcal{P}_l$ and $\mathcal{P}_r$, respectively, and generate $\mathcal{D}_{\mathcal{G}_o}$ by randomly picking b graphs from $\mathcal{D}_{\mathcal{G}_l} \cup \mathcal{D}_{\mathcal{G}_r}$. Then we compute $\mathbf{c}_o$ in line 8 where $F'_o \leftarrow \{f_r | f_r \in F, sup(f_r) \cap \mathcal{D}_{\mathcal{G}_o} \neq \emptyset\}$. The recursive process will terminate when $\mathcal{P}$ contains only 1 partition, and we can compute $\mathbf{c}$ by DSPM directly.

Now we discuss the partition method in Algorithm 7. According to Eq. (6), the computation of $\overline{x}_{ir}$ will not be affected if we omit the part $b_{ik}z_{kr}$ for all $g_k \notin sup(f_r)$. To reduce the cost on the update of $\overline{x}_{ir}$, we group the graphs with similar binary vectors into the same partition. This is processed in a recursive manner. We first sample n_o graphs from $\mathcal{D}_\mathcal{G}$, and cluster them into $\mathcal{O}_l$ and $\mathcal{O}_r$ as the center sets in line 4 using a classical clustering method such as k-means . Then we compute the graph-center distance for the rest of the graphs in $\mathcal{D}_\mathcal{G}$ and add them into the closer part. The graph-center distance between graph g_i and center set $\mathcal{O}$ is defined as $d(g_i, \mathcal{O}) = \sum_{g_j \in \mathcal{O}} \frac{d(\mathbf{y}_i, \mathbf{y}_j)}{|\mathcal{O}|}$. To balance the partition, we adjust

Algorithm 7 Partition $(\mathcal{D}_\mathcal{G}, F)$

1: **if** $|\mathcal{D}_\mathcal{G}| \leq b$ **then**
2: add $\mathcal{D}_\mathcal{G}$ to $\mathcal{P}$;
3: **else**
4: generate $\mathcal{O}_l$ and $\mathcal{O}_r$;
5: **for all** $g_i \in \mathcal{D}_\mathcal{G} \setminus (\mathcal{O}_l \cup \mathcal{O}_r)$ **do**
6: **if** $d(g_i, \mathcal{O}_l) \leq d(g_i, \mathcal{O}_r)$ **then**
7: $\mathcal{D}_{\mathcal{G}_l} \leftarrow \mathcal{D}_{\mathcal{G}_l} \cup \{g_i\}$;
8: **else**
9: $\mathcal{D}_{\mathcal{G}_r} \leftarrow \mathcal{D}_{\mathcal{G}_r} \cup \{g_i\}$;
10: adjust the size of $\mathcal{D}_{\mathcal{G}_l}$ and $\mathcal{D}_{\mathcal{G}_r}$ to balance;
11: Partition $(\mathcal{D}_{\mathcal{G}_l}, F)$;
12: Partition $(\mathcal{D}_{\mathcal{G}_r}, F)$;
13: **return** $\mathcal{P}$;

the graph numbers in these two parts in line 10. We set the number of graphs required in $\mathcal{D}_{\mathcal{G}_l}$ as $n_l = \lfloor \frac{n_p}{2} \rfloor \times b$. If $|\mathcal{D}_{\mathcal{G}_l}| > n_l$, we move $|\mathcal{D}_{\mathcal{G}_l}| - n_l$ graphs with the largest $d(g, \mathcal{O}_l)$ from $\mathcal{D}_{\mathcal{G}_l}$ to $\mathcal{D}_{\mathcal{G}_r}$; otherwise, we move $n_l - |\mathcal{D}_{\mathcal{G}_l}|$ graphs with the largest $d(g, \mathcal{O}_r)$ from $\mathcal{D}_{\mathcal{G}_r}$ to $\mathcal{D}_{\mathcal{G}_l}$. We stop the partition if the number of graphs in $\mathcal{D}_\mathcal{G}$ is no larger than the partition size b.

Theorem 5.3: *Algorithm 5 needs $O(k \cdot m' \cdot b \cdot n)$ time and $O(b \cdot (b + m'))$ memory space, where $m' \leq m$ is the maximum number of features contained in each partition.*

Proof: In Algorithm 5, both Partition and Computec are operations on binary tree structure. In each level of Partition, we need $O(n)$ time as n_o is usually very small and can be considered as a constant factor. Since the binary tree has $O(\log_2 \lceil \frac{n}{b} \rceil)$ levels, Partition needs $O(n \cdot \log_2 \lceil \frac{n}{b} \rceil)$ time overall. For Computec, in each node of the tree, we need to call function DSPM with time complexity $O(k \cdot b \cdot (m' \cdot b + b \cdot m')) = O(k \cdot m' \cdot b^2)$. The number of nodes in the tree is $2n_p - 1$ and Computec needs $O(k \cdot m' \cdot b^2 \cdot n_p) = O(k \cdot m' \cdot b \cdot n)$ time. The memory space is mainly determined by the function DSPM in the second phase, and it is $O(b \cdot (b + m'))$ as we need to store the configuration matrix $[\bar{x}_{ir}]_{b \times m'}$ and dissimilarity/distance matrix with $b \times b$ values. □

6. PERFORMANCE STUDIES

In this section, we show the effectiveness and efficiency of our approaches through extensive experiments. We compare our algorithms with seven algorithms: Original, Sample, SFS [21], MICI [24], MCFS [27], UDFS [28], and NDFS [29]. Original and Sample are two straightforward baselines where all the original frequent subgraphs and p randomly sampled frequent subgraphs are adopted as the dimensions, respectively. SFS sequentially selects the best p features, which is one of state-of-the-art greedy feature selection algorithms. MICI is a representative of the algorithms which select features based on feature similarity. MCFS, UDFS, and NDFS are the up-to-date methods in unsupervised feature selection to our best knowledge. We obtained the Matlab source code of MCFS and UDFS from the authors and implemented the rest of the algorithms except NDFS using Microsoft Visual C++. NDFS is implemented using Matlab since it involves the spectral analysis as MCFS and UDFS do. The tests were conducted on a 2.66GHz CPU and 3.43GB memory PC running Windows XP.

Datasets: We evaluate the performance of the algorithms on both real and synthetic datasets. The real chemical compound database is downloaded from the PubChem website. Each compound structure in this dataset can be modelled as an undirected labeled graph. We extract six datasets containing 1k, 2k, 4k, 6k, 8k, and 10k graphs, respectively. We also randomly extract another 1,000 graphs as the query set. Unless otherwise specified, we set the dataset with 1k graphs as the default dataset and use the other five datasets for scalability testing. The numbers of nodes in graphs range from 10 to 20. The frequent feature set F is mined by gSpan [38] with a minimum support 5%, which is a typical setting in frequent subgraph mining to generate a moderate number of frequent subgraphs. We generate a synthetic dataset with 1,000 database graphs and 1,000 query graphs using Graphgen [39]. The default parameters are set in a similar way as in [32], i.e., the average number of edges in each graph is 20, the number of distinct labels is 20, and the average graph density is 0.2. We further generate another ten synthetic datasets each with 1,000 database graphs and 1,000 query graphs to show the performance of the algorithms when varying the size and density of the graphs. Among them, five datasets are generated by varying the average number of edges from 12 to 20 with an average density of 0.2, and the other five datasets are generated by varying the density from 0.1 to 0.3 with an average edge number of 20.

Measures: Following the previous work in [2], we use Eq.(2) as δ to compute the graph dissimilarity in the experiment. The results of using Eq.(1) as δ are similar to those of using Eq.(2), thus are omitted in the paper due to space limit. For query q, we use $\mathcal{T}$ to represent the ranked list of the exact top-k results returned according to the graph dissimilarity, and use $\mathcal{A}$ to denote the ranked list of the approximate top-k results returned according to the graph distance in the feature space. To assess the effectiveness of the mapped feature space, we adopt the following widely-used measures [40] [41] [42] to evaluate the quality of the approximate top-k results.

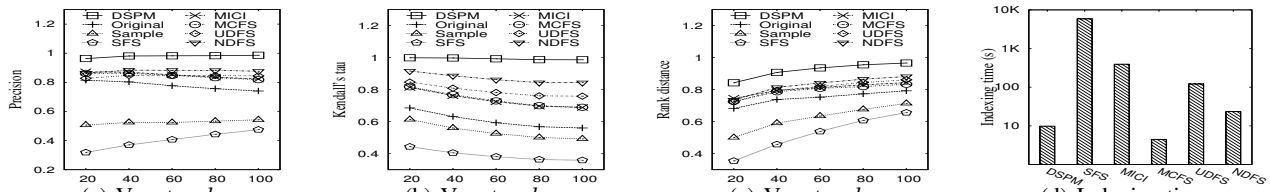
(1) *Precision*: the fraction of answers in the approximate top-k results that belong to the exact top-k results, which is defined as $p^{(k)} = |\mathcal{A} \cap \mathcal{T}|/k$.

(2) *Kendall's tau*: a measure to evaluate the correlation between two permutations [42]. Here, we adopt the definition which is modified to only evaluate the top-k members in the ranked list [40]: $\tau^{(k)} = \frac{\sum_{r_i \in \mathcal{A}} |\mathcal{A}_{i+1} \cap \mathcal{T}_{t(r_i)+1}|}{k(2n-k-1)}$, where r_i is the i-th element in the ranked list $\mathcal{A}$, $t(r_i)$ is the true rank of element r_i in $\mathcal{T}$, $\mathcal{A}_i$ is part of list $\mathcal{A}$ starting from the i-th element to the end, and $\mathcal{T}_{t(r_i)}$ is part of list $\mathcal{T}$ starting from the $t(r_i)$-th element to the end.

(3) *Rank distance*: the footrule distance between the ranks of members in the approximate top-k results and their true ranks in the exact top-k results, which is defined as $\gamma^{(k)} = \frac{\sum_{r_i \in \mathcal{A}} |i - t(r_i)|}{k}$ in [40] [41]. Here, we use the inverse form $\gamma^{(k)}_{inv} = \frac{k}{\sum_{r_i \in \mathcal{A}} |i - t(r_i)|}$ to make sure that $\gamma^{(k)}_{inv}$ is larger for result with better quality.

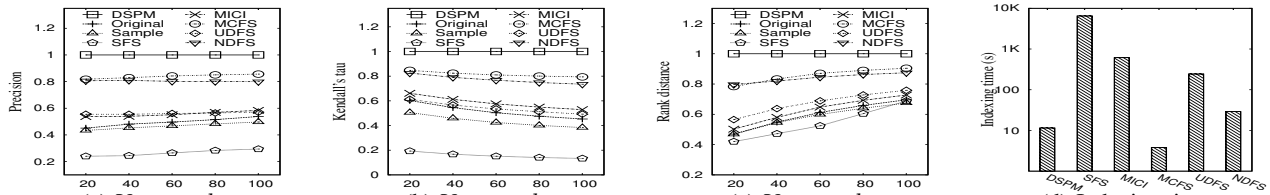
To the best of our knowledge, the optimal dimensional features of a graph database are unknown yet. Nevertheless, the experts in the chemical domain have provided a dictionary-based binary fingerprint for chemical structures in Pubchem dataset for similarity search. The similarity of two graphs is defined as the Tanimoto score of their fingerprints in PubChem. The fingerprint algorithm generates the top-k results based on the Tanimoto score. We consider the fingerprint algorithm as the benchmark and report the relative value for each evaluation measure, which is the ratio of the value achieved by each algorithm to the value achieved by the fingerprint algorithm. Since there is no fingerprint identified for the synthetic dataset, we use the best result generated among all these algorithms as the benchmark.

Parameters: We tune the parameters of MICI as suggested in paper [24]. For MCFS, UDFS, and NDFS, we adopt the default common parameter, 5, to specify the size of the neighborhoods, and tune the rest parameters by the way given in the corresponding papers. For DSPM, MICI, MCFS, UDFS, and NDFS, we report the best query result when we vary the number of selected features from {100, 200, 300, 400, 500} in a similar way to how it is done in [27] [28] [29]. For Sample and SFS, the number of selected features is set to the same as that in DSPM. In the following, we first compare the



(a) Vary top-k (b) Vary top-k (c) Vary top-k (d) Indexing time

Figure 4: Effectiveness Testing on Real Dataset



(a) Vary top-k (b) Vary top-k (c) Vary top-k (d) Indexing time

Figure 5: Effectiveness Testing on Synthetic Dataset (Vary Top-k)

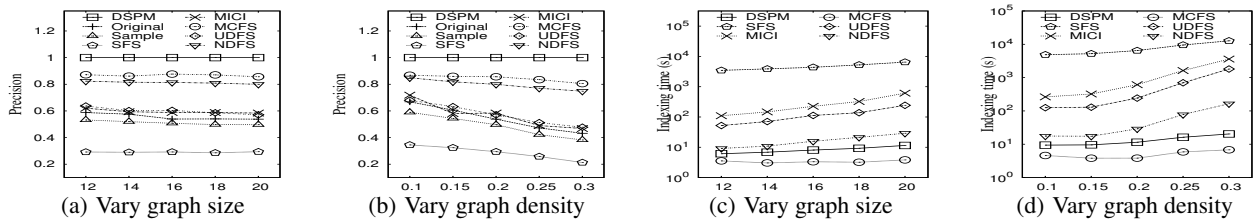
effectiveness of DSPM with the seven existing algorithms on both real and synthetic datasets, and then compare the performance of DSPM and DSPMap with other algorithms in terms of query efficiency, approximation quality, and scalability. For simplicity, we sequentially scan all vectors in the mapped multidimensional space to answer each query for all the algorithms, since the dominant cost for all algorithms is spent on computing the mappings.

(Exp-1) Effectiveness Testing on Real Dataset: We evaluate the performance of DSPM by comparing it with seven baseline algorithms on the real dataset, and report the results in Fig. 4. Fig. 4(a) shows the query precision for all the algorithms when we vary top-k. DSPM achieves the highest precision among all the algorithms and remains stable when k increases. The precisions for DSPM, MICI, MCFS, UDFS, and NDFS are higher than that of Original. This demonstrates that feature selection is very necessary for the frequent subgraph set to better preserve the dissimilarity and structure of graphs. The precision of Sample is very low compared with Original, which shows that random sampling is not a good choice for selecting dimensional features in graph databases. Surprisingly, we observe that SFS performs even worse than Sample. The underlying reason is that the objective function is non-monotonic which will causes SFS to get trapped in a local minimum far away from the optimal value. Fig. 4(b) shows the Kendall's tau for all the algorithms, where DSPM also outperforms all the baseline methods. There are large gaps between Original and NDFS, UDFS, MCFS, and MICI. The Kendall's tau for Original and that for Sample are both very low. SFS still has the worst performance among all the algorithms. Fig. 4(c) shows the rank distance for all the algorithms. DSPM still achieves the best result among all the algorithms, and the performance of other algorithms are similar as that in Fig. 4(a). In Fig. 4(d), we report the indexing time of these algorithms. The indexing time is the time of running the corresponding feature selection algorithm to choose the feature subset. Since there is no feature selection process in Original and Sample, we only report the indexing time for DSPM, SFS, MICI, MCFS, UDFS, and NDFS. As shown in the figure, both DSPM and MICI can select the features within 10 seconds. Note that although DSPM needs more time than MCFS, it in fact has the same worst time complexity as MCFS. SFS is the most expensive method among all the algorithms, because in each step to greedily select the next best feature, it has to compute the distance for all the graph pairs to obtain the value of objective function, which is very costly.

(Exp-2) Effectiveness Testing on Synthetic Dataset by Varying Top-k: We evaluate the performance of all the algorithms on synthetic dataset by varying top-k, and report the results in Fig. 5. Fig. 5(a) shows the query precision for all the algorithms when we vary top-k. DSPM still achieves the highest precision among all the algorithms and this is consistent with the result in Fig. 4(a). MICI, MCFS, UDFS, and NDFS perform better than Original, which shows that feature selection is also very necessary for synthetic dataset. Note that MCFS outperforms NDFS on synthetic data. This is different from the result in Fig. 4(a) in which NDFS outperform MCFS. This is because the performance of NDFS largely depends on the clustering characteristic of the dataset. The real chemical dataset usually has natural clusters/classes while the synthetic dataset generated by Graphgen does not have such characteristic. The performance of Original is almost as bad as Sample, which shows that the generated frequent subgraph is even more unbalanced for synthetic dataset compared with real dataset. SFS still performs the worst among all the algorithms. Fig. 5(b) shows the Kendall's tau for all the algorihtms, where DSPM outperforms all the baseline methods and SFS has the worst performance among all the algorithms. Fig. 5(c) shows the rank distance for all the algorithms. DSPM still achieves the best result among all the algorithms and the performance of other algorithms are similar as that in Fig. 5(a). Fig. 5(d) shows the indexing time of these algorithms on synthetic dataset. Both DSPM and MCFS are very fast and SFS is the most expensive among all the algorithms. Compared with the real dataset, all the algorithms have longer indexing time on synthetic dataset, given the fact that the number of frequent subgraphs mined in the synthetic dataset is larger than that in the real dataset.

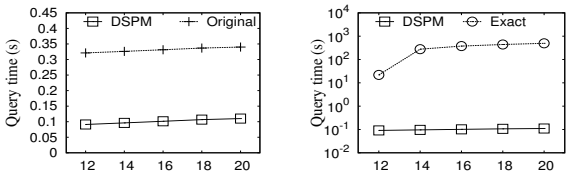
In the following, we use query precision as the effectiveness measure. The results for both Kendall's tau and rank distance are consistent with that in precision for each algorithm, and thus are omitted due to lack of space.

(Exp-3) Effectiveness Testing on Synthetic Dataset by Varying Graph Size and Density: We evaluate the performance of all the algorithms on synthetic dataset by varying the graph size and density, and show the results in Fig. 6. Fig. 6(a) shows the query precision for all the algorithms when we vary the graph sizes from 12 to 20. DSPM achieves the highest precision among all the algorithms while SFS performs the worst. MICI, MCFS, UDFS, and NDFS also perform better than Original which shows the necessity of feature selection no matter how the graph size varies. Compared with DSPM, the precision for other algorithms decreases slightly as the graph size increases. It's because more frequent subgraphs



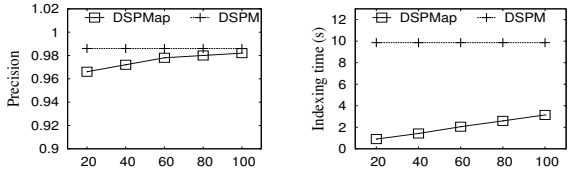
(a) Vary graph size (b) Vary graph density (c) Vary graph size (d) Vary graph density

Figure 6: Effectiveness Testing on Synthetic Dataset (Vary Graph Size and Density)



(a) Vary $|V(q)|$ (b) Vary $|V(q)|$

Figure 7: Query Efficiency Testing



(a) Vary b (b) Vary b

Figure 8: Approximation Quality Testing

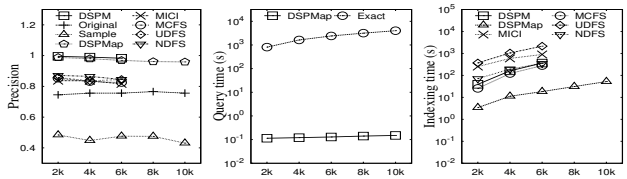
are generated when the graphs are larger and it will be harder to select good features from larger number of features for other algorithms. Fig. 6(b) shows the query precision for all the algorithms on synthetic dataset when we vary the graph density from 0.1 to 0.3. DSPM still achieves the best result among all the algorithms while SFS still performs the worst. Compared with DSPM, the precision for other algorithms also decreases when the graph size decreases. Note that the precision deceases more rapidly compared with the results in Fig. 6(a). It's because the number of frequent subgraphs increases more rapidly when the graphs become denser. In Fig. 6(c), we report the indexing time of these algorithms on synthetic dataset when we vary the graph size. MCFS and DSPM are always the fastest algorithms among these methods, and SFS is always the most expensive one. The indexing time for all the algorithms increases when the graph size increases. It' because more frequent subgraphs are generated from larger graphs. Note that the indexing time of DSPM and MCFS increases slower than that of UDFS, NDFS and MICI, due to the fact that the computation complexity of former two algorithms is linear with respect to number of features while the complexity of the latter three algorithms is at least quadratic with respect to the number of features. Fig. 6(d) shows the indexing time for synthetic dataset when we vary the graph density. The indexing time of all the algorithms increases when the graph density increases, because there are more frequent subgraphs generated in denser graphs as analyzed in Fig. 6(b).

In the following, we focus on real dataset. The curves for the synthetic dataset are similar to those on the real dataset for all the following tests, thus are omitted due to space limit.

(Exp-4) Query Efficiency Testing: We evaluate the query efficiency of the algorithms by varying the size of the query graph $|V(q)|$. We divide the query graphs into 5 groups: 10–12, 12–14, 14–16, 16–18, and 18–20 based on the value of $|V(q)|$. Fig. 7(a) shows the query time for DSPM and Original. For clarity, we do not show the query time for other algorithms as their query times are quite close to DSPM. This is because that they select similar number of features as DSPM does, and the query time largely depends on the number of features selected. The query time includes two parts: feature matching time and multidimensional search time. The former is to match each mapped feature with the query graph to generate a binary vector, which is done by the VF2 algorithm [43]. The latter is to retrieve the top-k result in the mapped feature space. The query time of Original is 3–5 times longer than that of DSPM because the number of mapped features, $|F|$, in Original is larger than the number of mapped features, p, in DSPM. As the size of query graph increases, the query time for both Original and DSPM increases slightly, because larger query graphs cost longer feature matching time by VF2. In Fig. 7(b), we compare DSPM with the exact algorithm which computes the exact top-k results based on the graph dissimilarity. The exact algorithm is very slow because it has to compute the MCS to obtain the graph dissimilarity. DSPM is 3–5 orders of magnitude faster than the exact algorithm.

(Exp-5) Approximation Quality Testing: We evaluate the performance of DSPMap regarding different partition sizes, and compare DSPMap with DSPM in terms of query precision and the indexing time. Fig. 8(a) shows the precision of DSPMap, which increases as the partition size increases from 20 to 100. The precision of DSPMap is almost the same as the precision of DSPM when the partition size is large, e.g., the gap between DSPMap and DSPM is less than 1% when the partition size is larger than 60. Even if the partition size is small, the precision of DSPMap is still very close to that of DSPM, e.g., the gap between DSPMap and DSPM is less than 2% when the partition size is reduced to 20. Fig. 8(b) shows the indexing time of DSPMap, which increases linearly as the partition size increases. This trend is consistent with the complexity analysis of DSPMap shown in Section 5. Considering both Fig. 8(a) and Fig. 8(b), when setting b to be 20, DSPMap loses 2% precision comparing to DSPM using 10 times faster indexing time.

(Exp-6) Scalability Testing: We evaluate the scalability of the DSPMap algorithm by comparing it with other algorithms when we vary the size of the graph database from 2k to 10k. We set the partition size to $b = \frac{|\mathcal{D}_\mathcal{G}|}{20}$. Fig. 9(a) shows the precision for all the algorithms except SFS as it cannot finish in 5 hours even for $|\mathcal{D}_\mathcal{G}| = 2k$. We can see that DSPMap constantly approximates DSPM well and outperforms all the other algorithms. MICI, MCFS, UDFS, NDFS, and DSPM cannot generate the results when the size of the graph database is larger than 6k as the data cannot fit into the PC memory. Fig. 9(b) shows the query time for DSPMap and the exact algorithm. For clarity, we did not plot the query time for other algorithms since they are very close to DSPMap. As the size of the graph database increases, the query time for both DSPMap and the exact algorithm increases, because we need to examine more graphs in a larger graph database. Overall, DSPMap is 3–5 orders of magnitude faster than the exact algorithm. Fig. 9(c) shows the indexing time for all the algorithms. DSPMap is the fastest and can deal with large graph databases very quickly. We did not obtain the indexing time for the rest of the algorithms when the graph database size is larger than 6k due to the



(a) Vary $|\mathcal{D}_\mathcal{G}|$ (b) Vary $|\mathcal{D}_\mathcal{G}|$ (c) Vary $|\mathcal{D}_\mathcal{G}|$

Figure 9: Scalability Testing

memory limit. Overall, DSPMap is 2–3 orders of magnitude faster than the other algorithms.

7. CONCLUSION

In this paper we study the *DS-preserved* mapping that preserves both distance and structure for online graph search. We show that the structure-preserving can be maintained up to a level, by showing a bound, if distance-preserving is achieved. We concentrate on identifying a small number of dimensions, where a dimension is a subgraph. We formulate the problem as a least square optimization problem and propose an iterative algorithm with optimization techniques. We also give an approximate algorithm to deal with large graph databases. The time complexity of the approximate algorithm linearly increases with the database size and it is scalable on large graph databases. Our experiments on real and synthetic datasets confirmed the effectiveness and efficiency of our approaches.

Acknowledgements. Yuanyuan Zhu was supported by FRFCU of China, No. 273363 and No.274172. Jeffrey Xu Yu was supported by grant of the RGC of Hong Kong SAR, No. CUHK 418512. Lu Qin was supported by ARC DE140100999.

8. REFERENCES

[1] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, vol. 19, no. 3-4, 1998.

[2] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng, "Finding top-k similar graphs in graph databases," in *Proc. of EDBT'12*, 2012.

[3] B. Luo, R. C. Wilson, and E. R. Hancock, "Spectral embedding of graphs," *Pattern Recognition*, vol. 36, no. 10, 2003.

[4] R. C. Wilson, E. R. Hancock, and B. Luo, "Pattern vectors from algebraic graph theory," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 7, 2005.

[5] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, 2000.

[6] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, 2000.

[7] D. Luo, C. H. Q. Ding, H. Huang, and T. Li, "Non-negative laplacian embedding," in *Proc. of ICDM'09*, 2009.

[8] T. Asano, P. Bose, P. Carmi, A. Maheshwari, C. Shu, M. Smid, and S. Wuhrer, "A linear-space algorithm for distance preserving graph embedding," *Computational Geometry*, vol. 42, no. 4, 2009.

[9] K. Riesen, M. Neuhaus, and H. Bunke, "Graph embedding in vector spaces by means of prototype selection," in *Graph-Based Representations in Pattern Recognition*, 2007.

[10] H. Bunke and K. Riesen, "Improving vector space embedding of graphs through feature selection algorithms," *Pattern Recognition*, vol. 44, no. 9, 2011.

[11] J. Gibert, E. Valveny, and H. Bunke, "Graph embedding in vector spaces by node attribute statistics," *Pattern Recognition*, vol. 45, no. 9, 2012.

[12] K. M. Borgwardt, N. N. Schraudolph, and S. Viswanathan, "Fast computation of graph kernels," in *Advances in Neural Information Processing Systems*, 2006.

[13] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. of ICDM'05*, 2005.

[14] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *International Workshop on Mining Graphs, Trees and Sequences*, 2003.

[15] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, 2007.

[16] X. Wang, A. M. Smalter, J. Huan, and G. H. Lushington, "G-hash: towards fast kernel-based similarity search in large graph databases," in *Proc. of EDBT'09*, 2009.

[17] M. Tan, F. Polat, and R. Alhajj, "Feature selection for graph kernels," in *IEEE International Conference on Bioinformatics and Biomedicine*, 2010.

[18] L. Schietgat, F. Costa, J. Ramon, and L. De Raedt, "Effective feature construction by maximum common subgraph sampling," *Machine Learning*, vol. 83, no. 2, 2011.

[19] L. Yu and H. Liu, "Efficiently handling feature redundancy in high-dimensional data," in *Proc. of KDD'03*, 2003.

[20] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1, 1997.

[21] R. Fukunaga, *Statistical Pattern Recognition*. Academic Press., 1990.

[22] L. Talavera, "Feature selection as a preprocessing step for hierarchical clustering," in *Proc. of ICML'99*, 1999.

[23] M. Dash, K. Choi, P. Scheuermann, and H. Liu, "Feature selection for clustering - a filter solution," in *Proc. of ICDM'02*, 2002.

[24] P. Mitra, C. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 3, 2002.

[25] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in Neural Information Processing Systems*, 2005.

[26] Z. Zhao and H. Liu, "Spectral feature selection for supervised and unsupervised learning," in *Proc. of ICML'07*, 2007.

[27] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data," in *Proc. of KDD'10*, 2010.

[28] Y. Yang, H. T. Shen, Z. Ma, Z. Huang, and X. Zhou, "l 2, 1-norm regularized discriminative feature selection for unsupervised learning," in *Proc. of IJCAI'11*, 2011.

[29] Z. Li, Y. Yang, J. Liu, X. Zhou, and H. Lu, "Unsupervised feature selection using nonnegative spectral analysis," in *Proc. of AAAI'12*, 2012.

[30] C. C. Aggarwal and H. Wang, *Managing and Mining Graph Data*. Springer, 2010, vol. 40.

[31] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proc. of SIGMOD'04*, 2004.

[32] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," in *Proc. of SIGMOD'07*, 2007.

[33] X. Yan, P. Yu, and J. Han, "Substructure similarity search in graph databases," in *Proc. of SIGMOD'05*, 2005.

[34] C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang, and X. Gu, "Towards graph containment search and indexing," in *Proc. of VLDB'07*, 2007.

[35] H. Cheng, X. Yan, J. Han, and C.-W. Hsu, "Discriminative frequent pattern analysis for effective classification," in *Proc. of ICDE'07*, 2007.

[36] J. De Leeuw, "Applications of convex analysis to multidimensional scaling," *Recent Developments in Statistics*, 1977.

[37] J. De Leeuw and H. W. J., "Multidimensional scaling with restrictions on the configuration," *Multivariate Analysis*, vol. V, 1980.

[38] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proc. of ICDM'02*, 2002.

[39] J. Cheng, Y. Ke, and W. Ng, "Graphgen: A graph synthetic generator," 2006, http://www.cse.ust.hk/graphgen.

[40] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 1, 2003.

[41] M. Theobald, G. Weikum, and R. Schenkel, "Top-k query evaluation with probabilistic guarantees," in *Proc. of VLDB'04*, 2004.

[42] M. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, 1938.

[43] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, 2004.