An Efficient Probabilistic Approach for Graph Similarity Search

Extended Technical Report

Zijian Li† Xun Jian‡ Xiang Lian† Lei Chen∗

†‡#Department of Computer Science and Engineering, HKUST, Hong Kong, China
∗Department of Computer Science, Kent State University, Kent, USA
{†zlicb, ‡xjian, #leichen}@cse.ust.hk  ‡xlian@kent.edu

ABSTRACT

Graph similarity search is a common and fundamental operation in graph databases. One of the most popular graph similarity measures is the Graph Edit Distance (GED) mainly because of its broad applicability and high interpretability. Despite its prevalence, exact GED computation is proved to be NP-hard, which could result in unsatisfactory computational efficiency on large graphs. However, exactly accurate search results are usually unnecessary for real-world applications especially when the responsiveness is far more important than the accuracy. Thus, in this paper, we propose a novel probabilistic approach to efficiently estimate GED, which is further leveraged for the graph similarity search. Specifically, we first take branches as elementary structures in graphs, and introduce a novel graph similarity measure by comparing branches between graphs, i.e., Graph Branch Distance (GBD), which can be efficiently calculated in polynomial time. Then, we formulate the relationship between GED and GBD by considering branch variations as the result ascribed to graph edit operations, and model this process by probabilistic approaches. By applying our model, the GED between any two graphs can be efficiently estimated by their GBD, and these estimations are finally utilized in the graph similarity search. Extensive experiments show that our approach has better accuracy, efficiency and scalability than other comparable methods in the graph similarity search over real and synthetic data sets.

1. INTRODUCTION

Graph similarity search is a common and fundamental operation in graph databases, which has widespread applications in various fields including bio-informatics, sociology, and chemical analysis, over the past few decades. For evaluating the similarity between graphs, Graph Edit Distance (GED) [1] is one of the most prevalent measures because of its wide applicability, that is, GED is capable of dealing with various kinds of graphs including directed and undirected graphs, labeled and unlabeled graphs, as well as simple graphs and multi-graphs (which could have multiple edges between two vertices). Furthermore, GED has high interpretability, since it corresponds to some sequences of concrete graph edit operations (including insertion of vertices and edges, etc.) of minimal lengths, rather than implicit graph embedding utilized in spectral [2] or kernel [3] measures. Example 1 illustrates the basic idea of GED.

Example 1. Assume that we have two graphs $G_1$ and $G_2$ as shown in Figure 1. The label sets of their vertices and edges are $\{A,B,C\}$ and $\{x,y,z\}$, respectively. The Graph Edit Distance (GED) between $G_1$ and $G_2$ is the minimal number of graph edit operations to transform $G_1$ into $G_2$. It can be proved that the GED between $G_1$ and $G_2$ is 3, which can be achieved by (1) deleting the edge between $v_1$ and $v_3$ in $G_1$, and (2) inserting an isolated vertex $v_3$ with label $A$, and (3) inserting an edge between $v_3$ and $v_4$ with label $x$.

With GED as the graph similarity measure, the graph similarity search problem is formally stated as follows.

Problem Statement: (Graph Similarity Search) Given a graph database $D$, a query graph $Q$, and a similarity threshold $\hat{\tau}$, the graph similarity search is to find a set of graphs $D_0 \subseteq D$, where the graph edit distance (GED) between $Q$ and each graph in $D_0$ is less than or equal to $\hat{\tau}$.

A straightforward solution to the problem above is to check exact GEDs for all pairs of $Q$ and graphs in database $D$. However, despite its prevalence, GED is proved to be NP-hard for exact calculations [1], which may lead to unsatisfactory computational efficiency when we conduct a similarity search over large graphs. The most widely-applied approach for computing exact GED is the $A^*$ algorithm [5], which aims to search out the optimal matching between the vertices of two graphs in a heuristic manner. Specifically, given two graphs with $n$ and $m$ vertices, respectively, the time complexity of $A^*$ algorithm is $O(n^m)$ in the worst case.

Previous works on improving the straightforward solution mainly follow the filter-and-verification framework [4] [6] [7] [8], which first filters out undesirable graphs from graph database $D$ by utilizing lower and upper bounds of GED, and then only calculates the GED between query graph $Q$ and each of those remaining candidates. Although these approaches can greatly accelerate the similarity search when the bounds are tight enough to filter out most graphs, the exact GEDs between the query graph and all candidate graphs still need to be computed during the verification step. Therefore, the efficiency of filter-and-verification approaches is restricted by the sizes of graphs.
due to the hardness of computing exact GED (NP-hard) [4] only conducted experiments on graphs with less than 10 vertices. In addition, a recent study [2] indicates that the $A^*$ algorithm is incapable of computing GED between graphs with more than 12 vertices, which can hardly satisfy the demand for searching real-world graphs. For instance, a common requirement in bio-informatics is to search and compare molecular structures of similar proteins [10]. However, the structures of human proteins usually contain hundreds of vertices (i.e., atoms) [11], which obviously makes similarity search beyond the capability of the approaches mentioned above. Another observation is that many real-world applications do not always require an exact similarity value, and an approximate one with some quality guarantee is also acceptable especially in real-time applications where the responsiveness is far more important than the accuracy. Taking the protein search as an example again, it is certainly more desirable for users to obtain an approximate solution within a second, rather than to wait for a couple of days to get the exact answer.

To address the problems above, many approaches have already been proposed to achieve an approximate GED between the query graph $Q$ and each graph in database $D$ in polynomial time [12], which can be leveraged to accelerate the graph similarity search by trading accuracy for efficiency. Assuming that there are two graphs with $n$ and $m$ vertices, respectively, where $n \geq m$, one well-studied method [13] [14] for estimating the GED between these two graphs is to solve a corresponding linear sum assignment problem, which requires at least $O(n^3)$ time for obtaining the global optimal value or $O(n^2 \log n^2)$ time for a local optimal value by applying the greedy algorithm [15]. An alternative method is spectral seriation [16], which first generates the vector representations of graphs by extracting their leading eigenvalues of the adjacency matrix ($O(n^2)$ time) [17], and then exploits a probabilistic model based on these vectors to estimate GED in $O(nm^2)$ time.

To further enhance the efficiency of GED estimation and better satisfy the demands for graph similarity search on large graphs, we propose a novel probabilistic approach which aims at estimating the GED with less time cost ($O(nd + \varepsilon^3)$), where $n$ is the number of vertices, $d$ is the average degree of the graphs involved, and $\varepsilon$ is the similarity threshold in the stated graph similarity search problem. Note that the similarity threshold $\varepsilon$ is often set as a small value (i.e., $\varepsilon \leq 10$) and does not increase with the number of vertices $n$ in previous studies [2] [8], thus, we can assume that $\varepsilon$ is a constant with regard to $n$ when the graph is sufficiently large. Moreover, most real-world graphs studied in related works [14] [15] are scale-free graphs [18], such as virus molecular, protein and antiviral compound structures, and we prove that the average degree $d = O(\log n)$ for scale-free graphs. Therefore, under the assumptions above, the time complexity of our approach is $O(n \log n + \varepsilon) = O(n \log n)$ for comparing one graph with the query graph, and $O(Dn \log n)$ for searching similar graphs in the graph database $D$, where $|D|$ is the number of graphs in database $D$.

To summarize, we have made the following contributions in this work.

- We adopt branches [8] as elementary structures in graphs, and define a novel graph similarity measure by comparing branches between graphs, i.e., Graph Branch Distance (GBD), which can be efficiently calculated in $O(nd)$ time.

- We build a probabilistic model which reveals the relationship between GED and GBD by considering branch variations as the result ascribed to graph edit operations. By applying our model, the GED between any two graphs can be estimated by their GBD in $O(\varepsilon^3)$ time.

- We conduct extensive experiments to show that our approach has better accuracy, efficiency and scalability compared with the related approximation methods over real and synthetic data sets.

The paper is organized as follows. In Section 2 we formally define the symbols and concepts which are used in this paper. In Section 3 we give definitions of branches, the branch isomorphism, and Graph Branch Distance (GBD). In Section 4 we introduce the extended graphs, which are exploited to simplify our model. In Section 5 we derive the probabilistic relation between GBD and GED, which is leveraged in Section 6 to perform the graph similarity search. In Section 7 we demonstrate the efficiency and effectiveness of our proposed approaches through extensive experiments. We discuss several related studies in Section 8. Finally, we conclude the paper in Section 9.

2. PRELIMINARIES

The graphs discussed in this paper are restricted to simple labeled undirected graphs. Specifically, the $i$-th graph in database $D$ is denoted by $G_i \equiv \{V_i, E_i, L\}$, where $V_i \equiv \{v_{i,1}, v_{i,2}, \ldots, v_{i,|V_i|}\}$ is the set of vertices, $E_i \equiv \{e_{i,1}, e_{i,2}, \ldots, e_{i,|E_i|}\}$ is the set of edges, while $L$ is a general labelling function. For any vertex $v_{i,j} \in V_i$, its label is given by $L(v_{i,j})$. Similarly, for any edge $e_{i,j} \in E_i$, its label is given by $L(e_{i,j})$. In addition, $L_V$ and $L_E$ are defined as the sets of all possible labels for vertices and edges, respectively. We also define $\varepsilon$ as a virtual label, which will be used later in our approach. When the label of a vertex (or edge) is $\varepsilon$, the vertex (or edge) is said to be virtual and does not actually exist. Particularly, we have $\varepsilon \notin L_V$ and $\varepsilon \notin L_E$.

In this paper, we take Graph Edit Distance (GED) [4] as the graph similarity measure, which is defined as follows.

DEFINITION 1 (Graph Edit Distance). The edit distance between graphs $G_1$ and $G_2$, denoted by $GED(G_1, G_2)$, is the minimal number of graph edit operations which are necessary to transform $G_1$ into $G_2$, where the graph edit operations (GEO) are restricted to the following six types:

- AV: Add one isolated labeled vertex;
- DV: Delete one isolated vertex;
- RV: Relabel one vertex;
- AE: Add one labeled edge;
- DE: Delete one edge;
- RE: Relabel one edge.
which transforms graph unique ID of this sequence. Then, according to Definition 1, \( v \) label of vertex all labels of edges adjacent to \( v \), branches, are pre-computed and stored with graphs. Suppose that all accessory data structures in different methods, \( \tau \) the similarity threshold

| Table 1: Table of Notations |
|-----------------------------|
| \( D \) ≜ \{ \( G_1, G_2, ..., G_{|D|} \) \}, the graph database |
| \( G_i \) ≜ \{ \( V_i, E_i, \mathcal{L} \) \}, \( i \)-th graph in database |
| \( V_i \) ≜ \{ \( v_{i,1}, v_{i,2}, ..., v_{i,|V_i|} \) \}, the vertices in \( G_i \) |
| \( E_i \) ≜ \{ \( e_{i,1}, e_{i,2}, ..., e_{i,|E_i|} \) \}, the edges in \( G_i \) |
| \( Q \) ≜ \{ \( V_Q, E_Q, \mathcal{L} \) \}, the query graph |
| \( \mathcal{L} \) ≜ labelling function for vertices and edges |
| \( \mathcal{L}_V \) ≜ the set of all possible vertex labels |
| \( \mathcal{L}_E \) ≜ the set of all possible edge labels |
| \( \varepsilon \) ≜ the virtual label |

3. BRANCH DISTANCE BETWEEN GRAPHS

To reduce the high cost of exact GED computations (NP-hard) in the graph similarity search, one widely-applied strategy for pruning search results is to exploit the differences between graph sub-structures as the bounds of exact GED values. In this paper, we consider the branches as elementary graph units, which are defined as:

**Definition 2 (Branches).** The branch rooted at vertex \( v \) is defined as \( B(v) = \{ \mathcal{L}(v), N(v) \} \), where \( \mathcal{L}(v) \) is the label of vertex \( v \), and \( N(v) \) is the sorted multiset containing all labels of edges adjacent to \( v \). The sorted multiset of all branches in \( G_i \) is denoted by \( B_{G_i} = \{ B(v) \mid v \in V_i \} \).

In practice, each branch \( B(v) \) is stored as a list of strings whose first element is \( \mathcal{L}(v) \) and the following elements are strings in the sorted multiset \( N(v) \). In addition, \( B_{G_i} \) for each graph \( G_i \) is stored in a sorted multiset, whose elements are essentially lists of strings (i.e., branches) and are always sorted ascendingly by the ordering algorithm. For a fair comparison of the computational efficiency, it is assumed that all accessory data structures are stored as lists of lengths \( d_v + 1 \) and \( d_u + 1 \), respectively. Therefore, checking whether \( B(v) \) and \( B(u) \) are isomorphic is essentially judging whether two lists of lengths \( d_v + 1 \) and \( d_u + 1 \) are identical, which can be done in \( O(1) \) time when \( d_v \neq d_u \) and otherwise in \( O(d_v) \) time. (Note that the length of a list can be obtained in \( O(1) \) time.)

Finally, we define the Graph Branch Distance (GBD).

**Definition 4 (Graph Branch Distance).** The branch distance between graphs \( G_1 \) and \( G_2 \), denoted by \( GBD(G_1, G_2) \), is defined as:

\[
GBD(G_1, G_2) = \max\{|B_{G_1} \cup B_{G_2}| - |B_{G_1} \cap B_{G_2}|, |V_1|, |V_2| - |B_{G_1} \cap B_{G_2}|
\]

where \( B_{G_1} \) and \( B_{G_2} \) are the multisets of all branches in graphs \( G_1 \) and \( G_2 \), respectively.

Example 3 below illustrates the process of computing GBD.

**Example 3.** Assume that we have two graphs \( G_1 \) and \( G_2 \) as shown in Figure 2. The branches rooted at the vertices in \( G_1 \) and \( G_2 \) are as follows:

\[
B(v_1) = \{ A; y, y \}, B(v_2) = \{ C; y, z \}, B(v_3) = \{ B; y, z \};
\]

\[
B(u_1) = \{ B; x, z \}, B(u_2) = \{ A; y \};
\]

\[
B(u_3) = \{ A; x \}, B(u_4) = \{ C; y, z \}.
\]

The sorted multisets of branches in \( G_1 \) and \( G_2 \) are:

\[
B_{G_1} = \{ B(v_1), B(v_2), B(v_3) \};
\]

\[
B_{G_2} = \{ B(u_1), B(u_2), B(u_3), B(u_4) \}.
\]

Finally, by multiset intersection, we can see that \( |B_{G_1} \cap B_{G_2}| = 1 \). Since \( \max\{|V_1|, |V_2|\} = 4 \), finally we have:

\[
GBD(G_1, G_2) = \max\{|V_1|, |V_2| - |B_{G_1} \cap B_{G_2}| = 3.
\]

It has been proved that the number of elements that need to be compared, when intersecting two sorted multisets, is \( \max\{m_1, m_2\} \). where \( m_1 \) and \( m_2 \) are the sizes of two multisets, respectively. Therefore, the GBD between query graph \( Q \) and any graph \( G \in D \) can be computed in time:

\[
\sum_v^n d_i = O(nd)
\]

where \( n = \max\{|V_Q|, |V_G|\} \), \( d_i \) is the degree of \( i \)-th compared vertex in \( G \), and \( d \) is the average degree of graph \( G \).

The GBD given by Definition 4 is utilized to model the graph edit process and further leveraged for estimating the graph edit distance (GED) in Section 5. The intuition behind our approach is as follows. The state-of-the-art method for pruning graph similarity search results assumes that the difference between branches of two graphs has a close relation to their GED. Therefore, we aim to use GBD to closely estimate the GED of two graphs.

4. EXTENDED GRAPHS

Before constructing our probabilistic model for estimating graph edit distances (GED) by branch distances (GBD), in this section, we reduce the number of graph edit operation (GEO) types that need to be considered, which helps to simplify our model. We show that, for any pair of graphs, by extending both graphs with virtual vertices and edges, we only need to consider two types of operations when modeling...
Moreover, according to Theorems 1 and 2, whenever the val-
convert graphs into their extended versions, and there is
between the extended graphs stay the same as the GED and
the graph edit process. Meanwhile, the GED and the GBD
k
k
G
in
phic to any other branches, we also have:
ations of types RV and RE when modeling the graph edit
process of transforming the extended graph \(G_1\) into \(G_2\).
Finally, given the graphs \(G_1\) and \(G_2\) (for \(|V_1| \leq |V_2|\), and their extended graphs \(G_1'\) and \(G_2'\), we have Theorems 1 and 2 which are utilized in the following sections.

**Theorem 1.** \(GED(G_1, G_2) = GED(G_1', G_2')\)

**Proof.** Please refer to 20. \(\square\)

**Theorem 2.** \(GBD(G_1, G_2) = GBD(G_1', G_2')\)

**Proof.** Let the sets of branches rooted at virtual vertices in \(G_1', G_2'\) be \(\Delta B_{G_1}\) and \(\Delta B_{G_2}\), respectively. We have:

\[ |B_{G_1} \cap B_{G_2}| = |B_{G_1} \cap B_{G_2} + \Delta B_{G_1} \cap B_{G_2}| + |B_{G_2} \cap \Delta B_{G_1} + \Delta B_{G_1} \cap B_{G_2}| \]

Since branches rooted at virtual vertices are not isomor-
phic to any other branches, we also have:

\[ |\Delta B_{G_1} \cap B_{G_2}| = |B_{G_1} \cap \Delta B_{G_2}| = |\Delta B_{G_1} \cap B_{G_2}| = 0 \]

Therefore, \(|B_{G_1}' \cap B_{G_2}'| = |B_{G_1} \cap B_{G_2}|\).

From the definitions of branches and extended graphs, and
\(|V_1| \leq |V_2|\), we have:

\[ max\{|V_1|, |V_2|\} = max\{|V_1'|, |V_2'|\} \]

From the definition of GBD, we can obtain:

\[ GBD(G_1, G_2) = max\{|V_1|, |V_2|\} - |B_{G_1} \cap B_{G_2}| \]

\[ = max\{|V_1'|, |V_2'|\} - |B_{G_1} \cap B_{G_2}'| = GBD(G_1', G_2') \]

That is, \(GBD(G_1, G_2) = GBD(G_1', G_2')\). \(\square\)

Note that the extended graph is only a conceptual model
for reducing the number of graph edit operation types that
need to be considered. In practice, we do not actually
convert graphs into their extended versions, and there is
no overhead for creating and maintaining extended graphs.
Moreover, according to Theorems 1 and 2 whenever the values
of \(GED(G_1', G_2')\) and \(GBD(G_1', G_2')\) are required, we can instead calculate \(GED(G_1, G_2)\) and \(GBD(G_1, G_2)\). That is, the calculations of GED and GBD are still conducted on original graphs rather than the extended ones in practice.

5. A PROBABILISTIC MODEL

In this section, we aim to solve the stated graph simi-
arity search problem by estimating GED from GBD in a
probabilistic manner. To be more specific, assuming that
we have two graphs \(G_1\) and \(G_2\) (for \(|V_1| \leq |V_2|\)), given

\(GBD(G_1, G_2) = \varphi\), the probability that \(GED(G_1, G_2) \leq \hat{\tau}\)
held can be calculated as follows.

\[
Pr[GED(G_1, G_2) \leq \hat{\tau} | GBD(G_1, G_2) = \varphi] = \sum_\tau Pr[GED(G_1, G_2) = \tau | GBD(G_1, G_2) = \varphi]
\]

\[
= \sum_\tau Pr[GED(G_1', G_2') = \tau | GBD(G_1', G_2') = \varphi]
\]

(3)

which can be proved by Theorems 1 and 2

For simplicity, in this section, we denote \(GED(G_1', G_2')\)
and \(GBD(G_1', G_2')\) by GBD and GBD respectively. By
applying Bayes’ Rule, we have

\[
Pr[GED = \tau | GBD = \varphi] = \Lambda_1(G_1', G_2'; \tau, \varphi) \cdot \Lambda_2(G_1', G_2'; \tau, \varphi)
\]

(4)

where

\[
\Lambda_1(G_1', G_2'; \tau, \varphi) = \frac{Pr[GBD = \varphi | GED = \tau]}{Pr[GBD = \varphi]}
\]

(5)

\[
\Lambda_2(G_1', G_2'; \tau, \varphi) = \frac{Pr[GED = \tau | GBD = \varphi]}{Pr[GED = \tau]}
\]

(6)

Therefore, the problem to estimate GED by GBD becomes
calculating \(\Lambda_1\) and \(\Lambda_2\), whose values are obtained in
the following two subsections.

5.1 Calculating \(\Lambda_1(G_1', G_2'; \tau, \varphi)\)

To compute \(\Lambda_1(G_1', G_2'; \tau, \varphi)\), in this subsection, we model
the procedure of a specific graph edit operation sequence as
a relabelling process whose result is some modifications
on branches, which is measured by branch distances between
dographs (GBD). Based-on our model, we prove Theorem 3
which gives the closed form of calculating \(\Lambda_1(G_1', G_2'; \tau, \varphi)\).

Specifically, we build our model by five steps as follows.

Step 1: We first consider GED as a random variable, whose
value is given when calculating \(\Lambda_1(G_1', G_2'; \tau, \varphi)\).

Step 2: We randomly choose one graph edit operation (GEO)
sequence from \(SEQ\) (i.e., the set of all possible GEO se-
quencies with the minimal length). We define this particular
choice as the random variable \(S\), whose value is the ID of
the chosen operation sequence. Note that observing \(GED = 3\)
is equivalent to giving \(SEQ\) where every \(seq \in SEQ\) satisfies
that \(|seq| = 3\), so \(S\) is dependent on \(GED\).

Definition 6. \(S(\omega)\) is the random variable where \(\omega\) is
a particular choice of operation sequence from \(SEQ\), and
\(S(\omega) = s\) if \(\omega\) choose the sequence with ID \(s\), that is, \(seq_s\).

Step 3: We model the numbers of RV and RE operations
in the sequence chosen in Step 2 as random variables \(X\) and
\(Y\), respectively.

Definition 7. \(X(s)\) is the random variable where \(s\) is a
particular value of \(S\), and \(X(s) = x\) if \(S(\omega) = s\) and the
number of operations with type RV in \(seq_s\) is \(x\).

Definition 8. \(Y(s)\) is the random variable where \(s\) is a
particular value of \(S\), and \(Y(s) = y\) if \(S(\omega) = s\) and the
number of operations with type RE in \(seq_s\) is \(y\). Note when
given \(GED = \tau\) and \(X = x\), we always have \(Y = \tau - x\).

Step 4: We define random variables \(Z\) as the number of
vertices covered by relabelled edges, and \(R\) as the number
of vertices either relabelled or covered by relabelled edges,
after conducting operations in the sequence chosen in Step 2.

Definition 9. \(Z(y)\) is the random variable where \(y\) is a
particular value of \(Y\), and \(Z(y) = m\) if \(Y(s) = y\) and
the number of vertices covered by relabeled edges is \(m\) when
conducting \(seq_s\).
**Theorem 3.**

\[
A_1(G_1, G_2; \tau, \varphi) = \Pr[GBD = \varphi \mid GED = \tau] = \frac{\Omega_1(x, \tau) \prod_{m} \Omega_2(m, x, \tau) \cdot \Omega_3(r, \varphi) \cdot \Omega_4(x, r, m)}{\Omega_5(x, r, m, \tau)}
\]

where

\[
\Omega_1(x, \tau) = H \left( x \mid |V_1^r| + \binom{|V_1^r|}{2}, |V_1^r|, \tau \right)
\]

\[
\Omega_2(m, x, \tau) = \binom{|V_1^r|}{2}^{-1} \sum_{t=0}^{m} (-1)^{m-t} \binom{|V_1^r|}{m} \binom{t}{r-2} - 1
\]

\[
\Omega_3(r, \varphi) = \binom{r}{r-\varphi} \cdot \frac{(2-\varphi)^2}{2^{r-\varphi}}
\]

\[
\Omega_4(x, r, m) = H(x + m - r \mid |V_1^r|, m, x)
\]

\[
\Omega_5(x, r, m, \tau) = \frac{1}{\mathcal{V}(\cdot) \cdot \mathcal{E}(\cdot)}
\]

**Proof.** Please refer to Appendix A.

### 5.2 Calculating the Prior Distribution of GBDs

In this subsection, we aim to calculate \(A_2(G_1, G_2; \tau, \varphi)\), which is essentially to infer the prior distributions of GEDs and GBDs respectively among all pairs of graphs involved in the graph similarity search. However, in most real-world scenarios, the query graph \(Q\) often comes from the same population as graphs in database \(D\). As a counter example, it is unusual to use a query graph of protein structure to search for similar graphs in social networks, and vice versa. Therefore, we assume that GEDs and GBDs between \(Q\) and each graph \(G\) in database \(D\), follow the same prior distributions as those among all graph pairs in \(D\).

To calculate the prior distribution of GBDs, we first randomly sample \(\alpha\%\) of graph pairs from the database \(D\), and then calculate the GBD between each pair of sampled graphs. Specifically, let \(|D|\) be the number of graph pairs in database \(D\), and then the number of sampled pairs of graphs is \(\alpha \% \cdot |D| \cdot (|D| - 1)\). Next, we utilize the Gaussian Mixture Model (GMM) \(\mathcal{G}\) to approximate the distribution of GBDs between all pairs of sampled graphs, which is a common approach to estimate unknown distribution densities. Finally, the probability density function (PDF) of GMM is:

\[
f(\varphi) = \sum_{i=1}^{K} \pi_i \cdot N(\varphi; \mu_i, \sigma_i)
\]

where \(K\) is the number of components in the mixture model, whose value is usually empirically set by user. \(N\) is the probability density function (PDF) of the normal distribution. Here, \(\pi_i, \mu_i, \text{ and } \sigma_i\) are parameters of the \(i\)-th component in the mixture model, which can be inferred from the GBDs over sampled graph pairs. Please refer to \(\mathcal{G}\) for the detailed process of inferring parameters in GMM.

Now, we can compute the prior probability \(\Pr[GBD = \varphi]\) by integrating the probability density function \(f(\varphi)\) on the adjacent interval of \(\varphi\), i.e., \(\varphi \in [0.5, 0.5 + 0.5]\).

\[
\Pr[GBD = \varphi] = \int_{-0.5}^{0.5} \sum_{i=1}^{K} \pi_i \cdot N(\varphi; \mu_i, \sigma_i) \, d\varphi
\]
In this section, first we elaborate on our graph similarity search algorithm (i.e., GDBA) based on the model derived in the previous section, which consists of two stages: offline pre-processing and online querying. Then, we study the time and space complexity of these two stages in detail.

Algorithm 1: Graph Similarity Search with Graph Branch Distance Approximation (GBDA)

**Input:** a query graph $Q$, a graph database $D$, a similarity threshold $\hat{\tau}$, and a probability threshold $\gamma$.

**Output:** the search result $D_0$

for each graph $G \in D$ do

**Step 1:** Pre-compute $A_2(Q',G';\tau,\varphi)$

**Step 2:** Calculate $GBD(Q',G')$ by Definition 4

**Step 3:** Given $GBD(Q',G') = \varphi$, we calculate $\Phi = Pr[GBD(Q,G) = \varphi]$

$= \sum_{\tau=0}^{\hat{\tau}} \sum_{\varphi=0}^{\max} A_1(Q',G';\tau,\varphi) \cdot A_2(Q',G';\tau,\varphi)$

where $A_1(Q',G';\tau,\varphi)$ is calculated by Theorem 3

**Step 4:** Insert $G$ into $D_0$ if $\Phi \geq \gamma$

end for

Assuming that we have a query graph $Q$, a graph database $D$, a similarity threshold $\hat{\tau}$, and a probability threshold $\gamma$, we can obtain the closed forms of $\frac{d}{d\tau} \Omega_1$ and $\frac{d}{d\tau} \Omega_2$ as follows.

$$\frac{d}{d\tau} \Omega_1(x,\tau) = \left( \frac{2}{\tau} \right)^{(v+1)}(n)^{\frac{1}{2}}(\tau-x)^{-1} \cdot F_1$$

$$\frac{d}{d\tau} \Omega_2(m, x, \tau) = \left( \frac{2}{\tau} \right)^{(v-1)}(n)^{-1}(m)^{\frac{1}{2}} \cdot F_2 \cdot \sum F_3 \cdot F_4$$

where

$$F_1 = H(\tau) - H\left( \frac{2}{\tau} (v+1) - 2\tau \right)$$

$$F_2 = \psi(\tau + 1) - \psi(x + 1 - \tau + \frac{1}{2}(v-1))$$

$$F_3 = (1-m)^{-\frac{1}{2}}(t-1)$$

and

$$F_4 = 1 + \psi(x + 1 - \tau + \frac{1}{2}(t-1)) - \psi(\tau + 1)$$

Note that this integration technique is commonly used in the field called continuity correction [23], which aims to approximate discrete distributions by continuous distributions. In addition, the choice of integral interval $[\varphi - 0.5, \varphi + 0.5]$ is a common practice in the continuity correction field [24].

We give Example 5 below to better illustrate the process of computing the prior distribution of GBDs.

**Example 5. In the Fingerprint data set of IAM Graph Database [29] containing 2,273 graphs (i.e., 5,164,256 pairs of graphs), we aim to calculate the prior distribution of GBDs of the graph pairs. We first randomly sample 60,000 pairs of graphs from the Fingerprint dataset, where the distribution of GBDs between all pairs of sampled graphs is represented by the blue histogram in Figure 4.

Then, we can infer the parameters of the Gaussian Mixture Model (GMM) from GBDs between all pairs of sampled graphs by utilizing the algorithm in [29]. The red line in Figure 4 shows inferred GMM probability density function, which is clearly close approximation to the real distribution of GBDs. This way, we can compute and store the probability $Pr[GBD = \varphi]$ for each possible value of $\varphi$ by Equation 14. The range of $\varphi$ is discussed in Section 6.3.

### 5.2.2 Calculating the prior distribution of GEDs

Now we focus on calculating the prior distribution of GEDs. Recall that the GED computation is NP-hard [4], and a recent study [9] even indicates that the most popular algorithm for calculating the exact GED is incapable of handling graphs with more than 12 vertices. Therefore, sampling some graph pairs and calculating the GED between each pair is infeasible especially for large graphs, which means that we cannot simply infer the prior distribution of GBDs from sampled graph pairs.

To address this problem, we utilized the Jeffreys prior [26] as the prior distribution of GEDs. The Jeffreys prior is well-known for its non-informative property [27], which means that it provides very little additional information to the probabilistic model. Therefore, Jeffreys prior is a common choice in Bayesian methods when we know little about the probabilistic model. Therefore, Jeffreys prior is a common choice in Bayesian methods when we know little about the actual prior distribution. By utilizing the Jeffreys prior, the prior probability $Pr[GED = \tau]$ is calculated as follows.

$$Pr[GED = \tau] = \frac{1}{C} \sum A_1 \cdot (\frac{d}{d\tau} \log A_1)^2$$

where $A_1$ is short for $A_1(G_1',G_2';\tau,\varphi)$, which is defined in Theorem 3. $\varphi$ is the value of GBD, and $C$ is a constant for normalization. From Equation 7, we have:

$$\frac{d}{d\tau} \log A_1 = \frac{1}{A_1} \left\{ \sum \Omega_1 \cdot \sum \Omega_2 \cdot \sum \Omega_3 \cdot \Omega_4 \right. + \sum \left( \frac{d}{d\tau} \Omega_1 \right) \sum \Omega_2 \cdot \sum \Omega_3 \cdot \Omega_4 \right\}$$

where $\Omega_1$, $\Omega_2$, $\Omega_3$ and $\Omega_4$ are short for $\Omega_1(x,\tau)$, $\Omega_2(m,x,\tau)$, $\Omega_3(r,\varphi)$ and $\Omega_4(x,r,m)$, respectively, which are defined in Theorem 3. After differentiating the right hand sides of Equations 8 and 9, we can obtain the closed forms of $\frac{d}{d\tau} \Omega_1$ and $\frac{d}{d\tau} \Omega_2$ as follows.

$$\frac{d}{d\tau} \Omega_1(x,\tau) = \left( \frac{2}{\tau} \right)^{(v+1)}(n)^{\frac{1}{2}}(\tau-x)^{-1} \cdot F_1$$

$$\frac{d}{d\tau} \Omega_2(m, x, \tau) = \left( \frac{2}{\tau} \right)^{(v-1)}(n)^{-1}(m)^{\frac{1}{2}} \cdot F_2 \cdot \sum F_3 \cdot F_4$$

which is clearly a close approximation to the real distribution of GEDs. The Jeffreys prior is well-known for its non-informative property [27], which means that it provides very little additional information to the probabilistic model. Therefore, Jeffreys prior is a common choice in Bayesian methods when we know little about the actual prior distribution. By utilizing the Jeffreys prior, the prior probability $Pr[GED = \tau]$ is calculated as follows.
the search results $D_0$ is achieved by the Algorithm 1 above, where $Q'$ and $G'$ are the extended graphs of $Q$ and $G$, respectively. Note that Step 1 tagged with symbol * can be conducted in the offline stage, as discussed in Section 5.2. We give the Example 6 below to better illustrate the process of each iteration in Algorithm 1.

**Example 6.** Assume that the graph $G_1$ in Figure 1 is the query graph $Q$, and $G_2$ in Figure 7 is a graph in database $D$. Given the similarity threshold $\tau = 3$ and the probability threshold $\gamma = 0.8$, the process of determining whether $G_2$ should be in the search result $D_0$ is as follows.

1. First, according to the methods described in Section 5.2, we can pre-compute the value of $\Lambda_2(Q', G_2'; \tau, \varphi)$ by inferring the prior distributions of GEDs and GBDs on database $D$. Since this is a simulated example and there is no concrete database $D$, we assume that $\Lambda_2(Q', G_2'; \tau, \varphi) \equiv 0.8$ for all possible values of $\tau$ and $\varphi$.

2. Second, from Example 3 we know GBD$(Q, G_2) = 3$.

3. Then, according to the formulae in Theorem 3 we have:

$$\Phi = \sum_{\tau=0}^{3} \Lambda_2(Q', G_2'; \tau, \varphi) \cdot \Lambda_2(Q', G_2'; \tau, \varphi) = (0 + 0 + 0.5113 + 0.5631) \times 0.8 = 0.8595 > \gamma = 0.8$$

Therefore, $G_2$ is inserted into the search result $D_0$.

### 6.2 Complexity Analysis of Online Stage

The online querying stage in our approach includes Steps 2, 3 and 4 in Algorithm 1 where Step 4 clearly costs $O(1)$ time for each graph $G$. In addition, from the discussions in Section 3, Step 2 costs $O(n d)$ time, where $n = \max\{|V_G|, |V_Q|\}$, and $d$ is the average degree of graph $G$.

Now we concentrate on analyzing the time complexity of computing Step 3 in Algorithm 1.

Since $A_2$ has already been pre-computed in the offline stage (i.e., Step 1), its value can be obtained in $O(1)$ time for each $\tau \in [0, \hat{\tau}]$ and graph $G \in D$.

Let the time for computing $\Omega_1, \Omega_2, \Omega_3$ and $\Omega_4$ in Theorem 3 be $C_1, C_2, C_3$ and $C_4$, respectively. From Equation (7) in Theorem 3 the time for computing $A_1$ for each $\tau \in [0, \hat{\tau}]$ and graph $G \in D$ is:

$$\sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} \Omega_2(x, m, \tau, \varphi) \cdot \Lambda_2(Q', G_2'; \tau, \varphi)$$

(25)

where $x, m$ and $\tau$ are the sumnation subscripts in Equation 7, and the ranges of $x, m$ and $\tau$ are:

- $x \in [0, \tau]$. Since $x$ is the number of RV operations, $x$ must not be larger than the number of graph edit operations $\tau$.
- $m \in [0, 2\tau]$. Since $m$ is the number of vertices covered by relabelled edges given the relabelled edge number $Y = \tau - x$, and each edge can cover at most two vertices, we have $0 \leq m \leq 2(\tau - x) \leq 2\tau$.
- $\tau \in [0, 3\tau]$. Note that $\tau$ is the number of vertices either relabeled or covered by relabeled edges when the relabelling vertex number is $X = x$ and the number of vertices covered by relabeled edges is $Z = m$. Therefore, we have $0 \leq \tau \leq x + m \leq \tau + 2\tau = 3\tau$.

Note that when utilizing Stirling’s Formula [29], we can calculate the combinational numbers in $O(1)$ time. Therefore, from Equations (6) ~ (11) in Theorem 3 it is clear that $C_1 = C_3 = C_4 = O(1)$ and $C_2 = O(m)$. Since $m \leq 2\tau$, we have $C_2 = O(\tau)$. Therefore, the time of computing $A_1$ for each $\tau \in [0, \hat{\tau}]$ and graph $G \in D$ is:

$$\sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} \Omega_2(x, m, \tau, \varphi) \cdot O(\tau) = O(\tau^3) + O(\tau^3) = O(\tau^3)$$

(26)

Moreover, from the above discussions about the ranges of summation subscripts, for any $\tau \in [0, \hat{\tau})$, we have:

$$\sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} \Omega_2(x, m, \tau, \varphi)$$

$$= \sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} P_r[Z = m | Y = \tau - x]$$

$$+ \sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} P_r[Z = m | Y = \tau - x]$$

$$+ \sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} P_r[Z = m | Y = \tau - x]$$

$$+ \sum_{x=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} P_r[Z = m | Y = \tau - x]$$

(27)

$$= f(m, x, \tau) + \sum_{\tau=0}^{2\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} \Omega_2(m, x, \tau)$$

(28)

where $f(m, x, \tau)$ is sum of last three terms in Equation (27).

Equation (28) means, the value of $\sum_{x, m} \Omega_2(m, x, \tau)$ where $\tau < \hat{\tau}$ have already been calculated in the process of computing $\sum_{x, m} \Omega_2(m, x, \hat{\tau})$. Therefore, we can reduce redundant computations by only computing $\sum_{x, m} \Omega_2(m, x, \tau)$ once to obtain values of $\sum_{x, m} \Omega_2(m, x, \tau)$ for all $\tau < \hat{\tau}$. Similar conclusions can also be derived for $\sum_{x, m} \Omega_2(x, \tau, \varphi)$, where the detailed proofs are omitted here.

So the time of computing Step 3 in Algorithm 1 is:

$$\sum_{\tau=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} \Omega_2(x, m, \tau, \varphi) \cdot O(\tau^3) + \sum_{\tau=0}^{\hat{\tau}} \sum_{m=0}^{2\hat{\tau}} \Omega_2(m, x, \tau) = O(\tau^3)$$

(29)

for each graph $G$ in database $D$.

In conclusion, the time of the whole online stage is:

$$O(n d) + O(\tau^3) + O(1) = O(n d + \tau^3)$$

(30)

for each graph $G$ in database $D$, where $n = \max\{|V_G|, |V_Q|\}$, $d$ is the average degree of graph $G$, and $\tau$ is the similarity threshold in the graph similarity search problem.

Note that the similarity threshold $\hat{\tau}$ is often set as a small value (i.e., $\hat{\tau} \leq 10$) and does not increase with the number of vertices $n$ in previous studies [4] [8], thus, we can assume that $\hat{\tau}$ is a constant with regard to $n$ when the graph is sufficiently large. Moreover, most real-world graphs studied in related works [13] [15] are scale-free graphs [13], such as virus molecular, protein and antivirus compound structures. The fraction of vertices with degree $k$ in scale-free graphs is $C \cdot k^{-\delta}$ [15], where $2 < \delta < 3$, and $C$ is a constant. Therefore, the average degree in scale-free graphs is:

$$d = \sum_{k=1}^{n} k \cdot C \cdot k^{-\delta} \cdot \frac{C}{k} = C \cdot H(n - 1) = O(\log n)$$

(31)

where $H(n)$ is the $n$-th Harmonic Number.

Therefore, under the assumptions above, the time of the whole online stage in our approach is:

$$O(n d + \hat{\tau}^3) = O(n \log n)$$

(32)

for each graph $G \in D$, and $O(|D| \log n)$ for the whole online stage, where $|D|$ is the graph number in database $D$.

### 6.3 Complexity Analysis of Offline Stage

The offline pre-processing stage in our approach is Step 1 in Algorithm 1 which is essentially to pre-compute the prior distributions of GEDs and GBDs respectively among all pairs of graphs involved in the graph similarity search.

#### 6.3.1 Complexity Analysis of Computing the Prior Distribution of GEDs

As discussed in Section 5.1, the prior distributions of GEDs can be pre-computed by the four steps below:

**Step 1.1:** Sample $\alpha\%$ of graph pairs from the database $D$.

**Step 1.2:** Calculate GBD between each sampled graph pairs.

**Step 1.3:** Learn the Gaussian Mixture Model (GMM) of the GBDs between sampled graph pairs.

**Step 1.4:** Calculate $Pr(GBD = \varphi)$ for each possible value of $\varphi$ by using Equation (15).
Let $N$ be the number of graph pairs sampled in Step 1.1. Then, Step 1.1 costs $O(N)$ time. From the discussions in Section 3, Step 1.2 costs $O(N \cdot nd)$ time, where $n$ is the maximal number of vertices among the sampled graphs, and $d$ is the average degree of the sampled graphs. The learning process of GMM in Step 1.3 costs $O(N \cdot K)$ time \cite{7}, where $K$ is the number of components in GMM, and $\epsilon$ is the maximal learning iterations for learning GMM.

As for Step 1.4, since $\varphi$ is the value of GBD, from the definition of GBD, the possible values of $\varphi$ are essentially $\{0, 1, 2, \ldots, n\}$, where $n$ is the maximal number of vertices among the sampled graphs. According to Equation (15), computing $Pr[GBD = \varphi]$ for each $\varphi$ values costs $O(K)$ time, where $K$ is the number of components in GMM derived in Step 1.3. Thus, Step 1.4 costs $O(nK)$ time.

Note that in the Gaussian Mixture Model, the component number $K$ and the maximal learning iterations $\epsilon$ are fixed constants. Therefore, the prior distributions of GBDs can be calculated in time $O(nK)$.

6.3.2 Complexity Analysis of Computing the Prior Distribution of GEDs

According to the discussions in Section 5.2, computing the prior distribution of GEDs is essentially calculating Equation (24) for each possible values of $\tau$ and $|V'_i|$. First, by comparing the structures of Equations (7) with (17), when $\tau$ and $|V'_i|$ are fixed values, it is clear that computing $\frac{\delta}{\phi} \log \Lambda_1$ costs the same time as computing $\Lambda_1$, which is $O(n^2)$, as discussed in Section 6.2, where $\delta$ is the user-defined similarity threshold. $n$ and $d$ are the maximal number of vertices and the average degree among all the graphs involved in the graph similarity search, respectively.

Then, it is clear that one graph edit operation can at most change two branches, the possible values of GBDs $\varphi$ in Equation (17) are essentially $\{0, 1, 2, \ldots, 2\tau\}$. According to Equation (16), given $\tau$ and $|V'_i|$, the prior probability value of $F(\tau, V'_i) = \tau$ can be calculated in time complexity $O(2\tau \cdot n^2) = O(n^3)$.

Finally, recall that computing the GED prior distribution is essentially calculating Equation (24) for all possible values of $\tau$ and $|V'_i|$, and it is clear that the possible values of $\tau$ are $\{0, 1, 2, \ldots, \tau\}$, where $\tau$ is the user-defined similarity threshold. In addition, the possible values of $|V'_i|$ are essentially $\{1, 2, \ldots, n\}$, where $n$ is the maximal number of vertices among all graphs involved in the graph similarity search. Therefore, the time of calculating the GED prior distribution is:

$$O(\tau \cdot n \cdot \tau^3) = O(n^2 \tau^3)$$

According to Section 5.2.2, we need to store a matrix whose rows represent possible values of $\tau$, and columns represent possible values of $|V'_i|$. Therefore, the space cost of storing the prior distribution of GEDs is $O(\tau \cdot n)$. Therefore, the time complexity of the offline stage is:

for $\tau \geq \hat{\tau}$:

$$O(Nnd) + O(n^3 \tau^3) = O(Nnd + n^3 \tau^3)$$

for $\tau < \hat{\tau}$:

$$O(Nnd) + O(n^3 \hat{\tau}^3) = O(Nnd + n^3 \hat{\tau}^3)$$

7. EXPERIMENTS

7.1 Data Sets and Settings

In this section, we evaluate our approaches on 3 real-world data sets (i.e., AIDS, Fingerprint and GREC) from the IAM Graph Database \cite{25}, and 2 synthetic data sets (i.e., Syn-1 and Syn-2). The statistics of data sets are listed in Table 2.

The 3 real-world data sets are widely-used for evaluating the performance of GED estimation methods in previous works \cite{14, 15, 16}. However, in order to evaluate how well the GED is approximated, we must know the exact value of GED, which is NP-hard to compute \cite{3}. Specifically, even the state-of-the-art method \cite{9} cannot compute one exact GED for graphs with 100 vertices within 48 hours on our machine. Thus, the sizes of graphs in the 4 real-world data sets are small, otherwise it is infeasible to calculate the exact GED for evaluations.

However, we still manage to evaluate our proposed method on large graphs. To address the problem above, we generate 2 sets of large random graphs (i.e., Syn-1 and Syn-2), where the GED between each pair of graphs is known. Both data sets Syn-1 and Syn-2 contain 7 subsets of graphs, where each subset contains 500 graphs whose numbers of vertices are 1K, 2K, 5K, 10K, 20K, 50K, 100K, respectively. The difference between data sets Syn-1 and Syn-2 is that in Syn-1 satisfy the scale-free property \cite{18} while graphs in Syn-2 are not, which means the fraction of vertices with degree $k$ in graphs of data set Syn-1 is proportional to $k^{-\delta}$ where $2 < \delta < 3$. The algorithm of generating synthetic graphs is described in Appendix C.

For each real data set, we randomly select 5% graphs as query graphs, while the remaining 95% graphs constitute the query database $D$. For each synthetic data set, we randomly select 10 graphs from each of its subset as query graphs.

We evaluate our method with the similarity threshold $\hat{\tau} = \{1, 2, \ldots, 10\}$, which is in the commonly used range of the similarity threshold in previous studies \cite{1, 8}. All our experiments are conducted on 12 machines where each of them has 32 Intel E5 CPUs (2-core, 2.40 GHz) and 128G DDR3 RAMs. The details of experiments will be described in the following sections.

7.2 Evaluating Offline Stage

In this section, we evaluate the time and space costs of the offline stage in our GBDA approach, which is essentially to pre-compute the prior distributions of GEDs and GBDs, on both real and synthetic data sets. In order to speed up the experiment progress, we calculate $F(\tau, v)$ for each possible

| Data Set | $|D|$ | $|Q|$ | $\nu_m$ | $\xi_m$ | $d$ | Scale-free |
|----------|------|------|-------|-------|-----|------------|
| AIDS     | 1896 | 100  | 95    | 103   | 2.1 | Yes        |
| Finger   | 2159 | 114  | 26    | 26    | 1.7 | Yes        |
| GREC     | 1045 | 55   | 24    | 29    | 2.1 | Yes        |
| Syn-1    | 3430 | 70   | 100K  | 1M    | 9.6 | Yes        |
| Syn-2    | 3430 | 70   | 100K  | 1M    | 9.4 | No         |

Note: $|D|$ is the number of graphs in database $D$, $|Q|$ is the number of query graphs. $\nu_m$ and $\xi_m$ are the maximal numbers of vertices and edges, respectively, while $d$ is the average degree. $K$ means thousand and $M$ means million.

and the space complexity of the offline stage is:

$O(n) + O(\hat{\tau} \cdot n) = O(n(1 + \hat{\tau}))$

where $n$ is the number of graph pairs sampled in Step 1.1, and $d$ is the maximal number of vertices and the average degree among all the graphs involved in the graph similarity search, respectively. $\hat{\tau}$ is the user-defined similarity threshold.

Table 2: Statistics of Data Sets
### Table 3: Costs of computing GBD prior distribution

| Data Set | AIDS | Finger | GREC | Syn-1 | Syn-2 |
|----------|------|--------|------|-------|-------|
| Time Costs | 11.1s | 7.5s   | 20.8s | 3.8h  | 3.2h  |
| Space Costs | 0.06kb | 0.04kb | 0.1kb | 0.3gb | 0.5gb |

### Table 4: Costs of computing GED prior distribution

| Data Set | AIDS | Finger | GREC | Syn-1 | Syn-2 |
|----------|------|--------|------|-------|-------|
| \(\sum_{i} T_i\) | 70.32h | 16.91h | 15.40h | 6.31h | 6.31h |
| \(\sum_{i} S_i\) | 1.9kb | 0.4kb | 0.4kb | 0.1kb | 0.1kb |
| max\{\(T_i\)\} | 0.3h | 0.3h | 0.3h | 0.4h | 0.4h |
| max\{\(S_i\)\} | 0.02kb | 0.02kb | 0.02kb | 0.01kb | 0.01kb |

Note: \(T_i\) and \(S_i\) are the time and space costs of the \(i\)-th parallel process, respectively. \(\sum_{i} T_i\) and \(\sum_{i} S_i\) are the overall time and space costs among all processes, respectively. max\{\(T_i\)\} and max\{\(S_i\)\} are the maximal time and space costs among all processes, respectively.

The value of \(\tau\) and \(v\) in parallel, where \(F\) is the probability mass function of the prior distribution of GEDs defined in Equation (21). In addition, the prior distribution of GBDs for each data set is calculated by a single process, respectively.

Table 3 shows the time and space costs of computing GBD prior distribution on different data sets, where the number of graph pairs sampled from each data set for estimating the prior distributions of GBDs is set to 100,000.

Let the time and space costs of the \(i\)-th parallel process be \(T_i\) and \(S_i\), respectively. Table 3 shows the overall time and space costs (i.e., \(\sum_{i} T_i\) and \(\sum_{i} S_i\)), as well as the maximal time and space costs (i.e., max\{\(T_i\)\} and max\{\(S_i\)\}) among all processes of computing the GBD prior distributions on different data sets, respectively. Particularly, since the processes of computing GBD prior distributions are fully parallelized, the actual time and space costs are max\{\(T_i\)\} and \(\sum_{i} S_i\), respectively. If we use only one process, the time and space costs will be \(\sum_{i} T_i\) and max\{\(S_i\)\}, respectively.

The experimental results show that, we can greatly reduce the time cost of calculating GED prior distribution with the support of parallel computing in practice. Moreover, since every pair of branches can be stored and compared separately from other pairs, we can also accelerate the computation of GBD prior distribution by calculating GBDs between sampled graphs in parallel.

In addition, we conduct experiments to analyze how the parameters in our method influence the time and space costs of the offline stage. Figures 14 and 15 show the trends of overall time and space costs for computing the prior distributions of GBDs and GEDs on the synthetic data set Syn-1, with various numbers of vertices in graphs (i.e., \(n\)), and different settings of parameters \(N\) and \(\tau\), respectively, where \(N\) is the number of graph pairs sampled from each data set for estimating the prior distributions of GBDs, and \(\tau\) is the user-defined similarity threshold.

The experimental results generally confirm the correctness of our complexity analysis in Section 6.3 of our paper. However, the overall time and space costs of computing the GED prior distribution do not exactly follow the theoretical analysis. This is because the possible numbers of vertices in synthetic graphs are essentially \(\{1K, 2K, 5K, 10K, ..., 100K\}\) instead of the worst-case range \(\{1, 2, 3, ..., 100K\}\) as claimed in Section 6.3. Therefore, the time complexity of computing GED prior distribution on synthetic data sets is \(O(C\tilde{\tau}^2)\) rather than \(O(n\tilde{\tau}^3)\), where \(C\) is a variable sub-linear to \(n\), and \(\tilde{\tau}\) is the similarity threshold. In addition, computing GED prior distributions on real data sets still costs \(O(n\tilde{\tau}^3)\) time since the number of vertices in their graphs ranges from 3 to \(n\), which explains why the overall time costs of computing GED prior distributions on the synthetic data sets (i.e., \(O(C\tilde{\tau}^2)\)) are smaller than the costs on most real data sets (i.e., \(O(n\tilde{\tau}^3)\)).

### 7.3 Evaluating Online Stage

In this subsection, we compare the efficiency and accuracy of the online stage of our GBDA approach with three competitors (i.e., the LSAP [14], Greedy-Sort-GED [15] and Graph Seriation [16]), by conducting graph similarity search tasks over both real and synthetic data sets. In the experiments, we also analyze how the efficiency and effectiveness (i.e., accuracy, recall and F1-score) of our method are influenced by the parameters, which are, the similarity threshold \(\tau\), the probability threshold \(\gamma\), and the number of vertices \(n\) in graphs (i.e., \(n\)).

#### 7.3.1 The Efficiency Evaluation

We first evaluate the query efficiency of our GBDA approach and three competitors by comparing the average query response time on each real data set with various similarity thresholds \(\tau\). The result in Figure 14 shows that our approach (i.e., GBDA) is more efficient than all the other three competitors on all real data sets when \(\tau\) is set to 1.5, and 10, respectively. The results also confirm the correctness of our complexity analysis in Section 6.2.

In addition, we studied how the number of vertices in graphs (i.e., \(n\)) influences the efficiency of our approach by comparing the query response time on synthetic data sets with various similarity thresholds \(\hat{\tau}\), where the results are shown in Figures 15 and 16.

The results in Figures 15 and 16 show that our approach is more efficient than the competitors on large graphs when the similarity threshold \(\hat{\tau}\) is 1, 2, 5, and 10. The results also show that the efficiency of our method is not influenced by whether the graphs are scale-free or not.

Note that the LSAP method can only handle graphs with less than 20K vertices while the other competitors (i.e., Greedy-Sort-GED and Seriation) can only handle graphs with 10K vertices. Specifically, when the graphs have more than 20K vertices, all of the competitors consume more than 128G memory on our machines, which exceeds the amount of our physical memory. However, our proposed method (i.e., GBDA) can handle graphs with 100K vertices efficiently, which means our method has better scalability (with respect to the number of vertices \(n\)) than the competitors.

#### 7.3.2 The Effectiveness Evaluation

We evaluate the effectiveness (i.e., accuracy, recall and F1-score) of our GBDA approach and three competitors by comparing the average precision of the query results on each real data set with various probability thresholds \(\gamma\). The results in Figures 17, 18, and 19 show that our approach always out-performs the other three competitors in accuracy when \(\gamma\) is set to 0.7, 0.8 and 0.9. Specifically, our method usually achieves the highest accuracy when \(\hat{\tau} = 2\), and slightly decreases when \(\hat{\tau}\) grows larger.

Although the recall of our method is lower than most competitors, as shown in Figures 19, 21 and 22, the F1-score of our method is mostly higher than the other three methods when \(\gamma\) is set to 0.7, 0.8 and 0.9. Specifically, our method usually achieves the highest recall and F1-Score when \(\hat{\tau} = 10\).

In addition, we studied how the number of vertices in graphs (i.e., \(n\)) influences the accuracy of our approach by...
comparing the precision of search results on synthetic data sets with various probability thresholds $\gamma$ and similarity thresholds $\hat{\tau}$. Recall that the LSAP method can at most handle graphs with less than 20K vertices while the other competitors (i.e., Greedy-Sort-GED and Seriation) can only handle graphs with 10K vertices on our machine. From the results shown in Figures 26–29, it is clear that the accuracy of our method is stable to the graph size (i.e., the number of vertices). There is no significant difference between the accuracy of our method under various settings of the probability threshold $\gamma$. This results demonstrate the robustness of our method under different settings of parameters.

8. RELATED WORKS

8.1 Exact GED Computation

The $A^*$ algorithm [5] is the state-of-the-art method for exact GED computation, which searches the whole space of possible bijective mappings between the vertices of two graphs, and finds the mapping with the minimal edit cost by a best-first search. Recently, a method [9] was proposed as an improvement to the $A^*$ algorithm on sparse graphs, which aims to find the optimal matching between edges rather than vertices. The exact GED computation can also be formulated as a quadratic assignment problem (QAP) [30], which explicitly represents the graph edit cost deduced from a particular vertex mapping by the sum of costs for all vertex and edge edit operations, respectively. It then searches the vertex mapping with minimal edit cost in an optimization process. It has been proved that the time for calculating the exact GED is exponential in terms of the sizes of the involved graphs [4], which restricts the efficiency of graph similarity searches on large graphs. A prevalent improvement of the graph similarity search based on exact GED is to apply the filter-and-verification framework [4] [6] [7] [8], which first filters out undesirable graphs from the graph database by utilizing lower and upper bounds of GED, and then only calculates the exact GED between the query graph and each of those remaining candidates.

8.2 GED Estimation

Due to the hardness of computing the exact GED (NP-hard) [4], various approaches have been proposed to achieve an approximate GED by trading accuracy for efficiency.

One well-studied method [13] [14] for estimating the GED between two graphs is to solve a corresponding linear sum assignment problem (LSAP), which can be achieved by simplifying the objective function in the quadratic assignment problem (QAP) [30], and is essentially equivalent to the bipartite graph matching problem [14]. Such approaches formulate the graph similarity search problem as an optimization problem, and obtain the search results by optimization techniques such as the Hungarian method [31], the greedy method [15] and the genetic algorithm [32]. Therefore, the basic idea and the formulation of the graph similarity search problem of these approaches are entirely different from ours.

Another state-of-the-art approach is graph seriation [16], which first converts graphs into one-dimensional vectors by extracting their leading eigenvalues of the adjacency matrix, and then exploits a probabilistic model based on these vectors to estimate the GED. Although both this approach and ours utilize probabilistic models, the structure of our model is totally different from the prior work [16]. In addition, their model takes the leading eigenvalues of the adjacency matrix as the inputs, while the inputs of our model are the GBDs.

There are also methods which introduce machine learning approaches to estimate the GED by support vector regression (SVR) [33] and artificial neural networks [34]. Note that, it is NP-hard [4] to obtain the exact GEDs, so these supervised approaches can hardly learn their models from the exact GED values directly. Thus, these approaches usually take GED estimations from other methods as inputs, and are used to enhance the accuracy of other methods [33].
9. CONCLUSIONS

In this paper, we define the branch distance between two graphs (GBD), and further prove that the GBD has a probabilistic relationship with the GED by considering branch variations as the result ascribed to graph edit operations and modeling this process by probabilistic approaches. Furthermore, this relation between GED and GBD is leveraged to perform graph similarity searches. Experimental results demonstrate both the correctness and effectiveness of our approach, which outperforms the comparable methods. However, our approach can still be further extended in the following directions.

- Since every branch can be stored separately from others, and branch comparisons can be conducted in parallel, it is convenient to migrate our data structures and algorithms onto a vertex-central distributed graph processing system, which deserves further investigation.
• One very specific kind of sub-structures is chosen as the basic unit in our model, that is, the branch. However, by applying our modeling approach, it is also interesting to build new probabilistic models based on other graph sub-structures.

10. REFERENCES

[1] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. IEEE transactions on systems, man, and cybernetics, (3):353–362, 1983.

[2] Terry Caelli and Serhiy Kosinov. An eigenspace projection clustering method for inexact graph matching. IEEE transactions on pattern analysis and machine intelligence, 26(4):515–519, 2004.

[3] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In Learning Theory and Kernel Machines, pages 129–143. Springer, 2003.

[4] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: on approximating graph edit distance. Proceedings of the VLDB Endowment, 2(1):25–36, 2009.

[5] Peter E Hart, Nilis J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

[6] Guoren Wang, Bin Wang, Xiaochun Yang, and Ge Yu. Efficiently indexing large sparse graphs for similarity search. IEEE Transactions on Knowledge and Data Engineering, 24(3):440–451, 2012.

[7] Xiang Zhao, Chuan Xiao, Xueming Lin, and Wei Wang. Efficient graph similarity joins with edit distance constraints. In IEEE 28th International Conference on Data Engineering, pages 834–845. IEEE, 2012.

[8] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Graph similarity search with edit distance constraint in large graph databases. In CIKM ’13: Proceedings of the 22nd ACM international conference on Information & Knowledge Management, 2013.

[9] Karam Gouda and Mosab Hassaan. Cs_ged: An efficient approach for graph edit similarity computation. In 32nd IEEE International Conference on Data Engineering, ICDE, pages 265–276, 2016.

[10] Rashid Ibragimov, Maximiliane Malek, Jiong Guo, and Jan Baumbach. Gedevol: an evolutionary graph edit distance algorithm for biological network alignment. In OASics-OpenAccess Series in Informatics, volume 34. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[11] Rob Phillips Ron Milo. Cell Biology by the Numbers. Garland Science, 2015.

[12] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. Pattern Analysis and applications, 13(1):113–129, 2010.

[13] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoît Gariéry, and Mario Vento. Graph edit distance as a quadratic assignment problem. Pattern Recognition Letters, 2016.

[14] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision computing, 27(7):950–959, 2009.

[15] Kaspar Riesen, Miquel Ferrer, and Horst Bunke. Approximate graph edit distance in quadratic time. IEEE/ACM transactions on computational biology and bioinformatics, 2015.

[16] Antonio Robles-Kelly and Edwin R Hancock. Graph edit distance from spectral seriation. IEEE transactions on pattern analysis and machine intelligence, 27(3):365–378, 2005.

[17] Gene H Golub and Charles F Van Loan. Matrix computations, volume 3. JHU Press, 2012.

[18] Wikipedia. Scale-free network, 2017. https://en.wikipedia.org/w/index.php?title=Scale-free_network

[19] C++ Standard Library. std::lexicographical_compare, 2017. http://www.cplusplus.com/reference/algorithm/lexicographical_compare/

[20] Derek Justice and Alfred Hero. A binary linear programming formulation of the graph edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(8):1200–1214, 2006.

[21] Francesc Serratosa. Fast computation of bipartite graph matching. Pattern Recognition Letters, 45:244–250, 2014.

[22] Neil E Day. Estimating the components of a mixture of normal distributions. Biometrika, 56(3):463–474, 1969.

[23] Wikipedia. Continuity correction - wikipedia, 2017.

[24] Morris H DeGroot and Mark J Schervish. Probability and Statistics: Pearson New International Edition. Pearson Higher Ed, 2013.

[25] Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. Structural, Syntactic, and Statistical Pattern Recognition, pages 287–297, 2008.

[26] Harold Jeffreys. An invariant form for the prior probability in estimation problems. In Proceedings of the Royal Society of London a: mathematical, physical and engineering sciences, volume 186, pages 453–461. The Royal Society, 1946.

[27] Wikipedia. Jeffrey’s prior - wikipedia, 2017.

[28] Wikipedia. Digamma function - wikipedia, 2017.

[29] Lucien Le Cam. The central limit theorem around 1935. Statistical science, pages 78–91, 1986.

[30] Kaspar Riesen, Andreas Fischer, and Horst Bunke. Computing upper and lower bounds of graph edit distance in cubic time. In IAPR Workshop on Artificial Neural Networks in Pattern Recognition, pages 129–140. Springer, 2014.

[31] Harold W Kuhn. The hungarian method for the assignment problem. Naval research logistics quarterly, 2(1-2):83–97, 1955.

[32] Kaspar Riesen, Andreas Fischer, and Horst Bunke. Improving approximate graph edit distance using genetic algorithms. In Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition, pages 63–72. Springer, 2014.

[33] Kaspar Riesen. Learning exact graph edit distance. In Structural Pattern Recognition with Graph Edit
\[ \Omega = \Pr[\text{GED} = \varphi \mid \text{GED} = \tau] \]  
(34)

By marginalizing out \( S \) from \( \Omega \):
\[ \Omega = \sum_s \{ \Pr[\text{GED} = \varphi, S = s \mid \text{GED} = \tau] \} \]

By applying Chain Rule on \( \Omega \):
\[ \Omega = \sum_s \{ \Pr[\text{GED} = \varphi \mid S = s, \text{GED} = \tau] \cdot \Pr[S = s \mid \text{GED} = \tau] \} \]

In our model, every GED operation sequence is randomly selected with same probability, therefore,
\[ \Pr[S = s \mid \text{GED} = \tau] = 1/|\text{SEQ}| \]

Let \( N = |\text{SEQ}| \) and we have:
\[ \Omega = \frac{1}{N} \sum_s \{ \Pr[\text{GED} = \varphi, S = s, \text{GED} = \tau] \} \]

By marginalizing out \( X \) and \( Y \) from \( \Omega \):
\[ \Omega = \frac{1}{N} \sum_{s,x,y} \{ \Pr[\text{GED} = \varphi, X = x, Y = y \mid S = s, \text{GED} = \tau] \} \]

Since \( Y = \tau - x \) when given \( \text{GED} = \tau \) and \( X = x \):
\[ \Omega = \frac{1}{N} \sum_{s,x} \{ \Pr[\text{GED} = \varphi, X = x, Y = \tau - x \mid S = s, \text{GED} = \tau] \} \]

By applying Chain Rule on \( \Omega \):
\[ \Omega = \frac{1}{N} \sum_{s,x} \sum_x \{ \Pr[\text{GED} = \varphi \mid X = x, Y = \tau - x, S = s, \text{GED} = \tau] \cdot \Pr[X = x, Y = \tau - x \mid S = s, \text{GED} = \tau] \} \]

According to the Bayesian Network in Figure 2, we have:
\[ \Omega = \frac{1}{N} \sum_s \sum_x \{ \Pr[\text{GED} = \varphi \mid X = x, Y = \tau - x] \cdot \Pr[X = x, Y = \tau - x \mid S = s] \} \]

Define:
\[ \Theta_1(x, \tau) = \Pr[\text{GED} = \varphi \mid X = x, Y = \tau - x] \]  
(35)
\[ \Omega_1(x, \tau) = \frac{1}{N} \sum_s \Pr[X = x, Y = \tau - x \mid S = s] \]  
(36)

Then:
\[ \Omega = \sum_x \{ \Omega_1(x, \tau) \cdot \Theta_1(x, \tau) \} \]  
(37)

Likewise, by marginalizing out \( Z \) from \( \Theta_1(x, \tau) \) and applying Chain Rule:
\[ \Theta_1(x, \tau) = \sum_m \{ \Pr[\text{GED} = \varphi \mid Z = m, X = x, Y = \tau - x] \cdot \Pr[Z = m \mid X = x, Y = \tau - x] \} \]

From Bayesian Network in Figure 2 we have:
\[ \Theta_1(x, \tau) = \sum_m \{ \Pr[\text{GED} = \varphi \mid X = x, Z = m] \cdot \Pr[Z = m \mid Y = \tau - x] \} \]

Define:
\[ \Omega_2(m, x, \tau) = \Pr[Z = m \mid Y = \tau - x] \]  
(38)
\[ \Theta_2(m, x, \varphi) = \Pr[\text{GED} = \varphi \mid X = x, Z = m] \]  
(39)

We have:
\[ \Theta_1(x, \tau, \varphi) = \sum_m \{ \Omega_2(m, x, \tau) \cdot \Theta_2(m, x, \varphi) \} \]  
(40)

Likewise, by marginalizing out \( R \) from \( \Theta_2(m, x, \varphi) \) and then applying Chain Rule:
\[ \Theta_2(m, x, \varphi) = \sum_r \{ \Pr[\text{GED} = \varphi \mid R = r, X = x, Z = m] \cdot \Pr[R = r \mid X = x, Z = m] \} \]

From Bayesian Network in Figure 2 we have:
\[ \Theta_2(m, x, \varphi) = \sum_r \{ \Pr[\text{GED} = \varphi \mid R = r] \cdot \Pr[R = r \mid X = x, Z = m] \} \]

Finally, from Equations 34 and 12 we can obtain:
\[ \Omega = \sum_x \Omega_1(x, \tau) \cdot \Theta_1(x, \tau) \]  
= \sum_x \Omega_1(x, \tau) \cdot \sum_m \Omega_2(m, x, \tau) \cdot \Theta_2(m, x, \varphi) \]  
= \sum_x \Omega_1(x, \tau) \cdot \sum_m \Omega_2(m, x, \tau) \cdot \sum_r \Theta_3(r, \varphi) \cdot \Theta_4(x, r, m) \]

Therefore, Theorem 3 is proved, while the formulae for calculating \( \Omega_1, \Omega_2, \Omega_3 \) and \( \Omega_4 \) are given in Lemmas 1, 2, 3 and 4 respectively. Please refer to Appendices B, C, D and E for Lemmas 1, 2, 3 and 4 respectively.

B. LEMMA 1 AND ITS PROOF

**Lemma 1.** Given \( \text{GED} = \tau \), for any integer \( x \in [0, \tau] \), we have:
\[ \Omega_1(x, \tau) = \sum_{s \in \text{SEQ}} \Pr[X = x, Y = \tau - x \mid S = s] = \mathcal{H}(x; |V'_1| + ([V'_1]_x), |V'_1|, \tau) \]

where function \( \mathcal{H}(x; M, K, N) \) is defined in Equation 12.

**Proof.** From the definitions, \( \Omega_1(x, \tau) \) is the probability of a random graph edit sequence \( \text{seq} \) exactly relabelling \( x \) vertices and \( \tau - x \) edges. Since the extended graph \( G'_1 \) is a complete graph, it has \( |E'_1| = ([V'_1]_x) \) edges. Therefore, the number of ways to choose \( x \) vertices for relabelling is \( (\binom{|V'_1|}{x}) \), and the number of ways to choose \( \tau - x \) edges for relabelling is \( \binom{|V'_1|}{\tau - x} \). Then we have:
\[ \Omega_1(x, \tau) = \frac{(\binom{|V'_1|}{x}) \cdot (\binom{|V'_1|}{\tau - x})}{(\binom{|V'_1|}{x} + (\binom{|V'_1|}{\tau - x}))} = \mathcal{H}(x; |V'_1| + ([V'_1]_x), |V'_1|, \tau) \]

where function \( \mathcal{H}(x; M, K, N) \) is defined in Equation 12. □
C. LEMMA 2 AND ITS PROOF

LEMMA 2. Given $GED = \tau$, for any integer $x \in [0, \tau]$ and $m \in [0, \lvert V_1' \rvert]$, we have:

$$\Omega_2(m, x, \tau) = Pr\{ Z = m \mid Y = \tau - x \} = \left( \frac{\binom{\lvert V_1' \rvert}{m}}{\binom{\lvert V_1' \rvert}{x}} \right) \sum_{t=0}^{m} (-1)^{m-t} \binom{\lvert V_1' \rvert}{m} \binom{\lvert V_1' \rvert - t}{x} \binom{x}{t} \binom{\tau - x}{m - t}$$

(44)

Proof. Let $x' = \tau - x$. Since the extended graph $G'_1$ is a complete graph, it has $\lvert E'_1 \rvert = \binom{\lvert V_1' \rvert}{2}$ edges. From definitions of $Y$ and $Z$, $\Omega_2(m, x)$ can be modelled by following problem:

- Randomly select $x'$ edges from a complete graph with $\lvert V_1' \rvert$ vertices and $(\binom{\lvert V_1' \rvert}{2})$ edges, what is the probability of these edges exactly covering $m$ vertices?

Let $V_X$ be the vertices covered by an edge subset $X \subseteq E'_1$. By the inclusion-exclusion principle, for any vertex subset $S \subseteq V_1'$ where $\lvert S \rvert = m$, the number of possible edge sets $X$ which satisfies $V_X = S$ is:

$$k_{x',m} = \sum_{t=0}^{m} (-1)^{m-t} \binom{\lvert V_1' \rvert}{t} \binom{x}{t} \binom{\tau - x}{m - t}$$

Since the number of sets $S \subseteq V_1'$ of size $m$ is $\binom{\lvert V_1' \rvert}{m}$, the total number of ways to pick an edge set $X$ which satisfies $\lvert X \rvert = x'$ and $\lvert V_X \rvert = m$ is:

$$K_{x',m} = \sum_{t=0}^{m} (-1)^{m-t} \binom{\lvert V_1' \rvert}{m} \binom{\lvert V_1' \rvert - t}{x'} \binom{x}{t} \binom{\tau - x}{m - t}$$

Also the number of ways to pick an edge set $X \subseteq E'_1$ is:

$$K_{x'} = \sum_{m} K_{x',m} = \binom{\lvert V_1' \rvert}{x'}$$

Therefore, we have:

$$\Omega_2(m, x, \tau) = \frac{K_{x',m}}{K_{x'}} = \frac{k_{x',m}}{\binom{\lvert V_1' \rvert}{x'}}$$

So Lemma 2 is proved. □

D. LEMMA 3 AND ITS PROOF

LEMMA 3. Given $GBD = \varphi$, for any integer $r \in [0, \lvert V_1' \rvert]$, we have:

$$\Omega_3(r, \varphi) = Pr\{ GBD = \varphi \mid R = r \} = \binom{r}{r - \varphi} \cdot \frac{(\varphi - 1)^{x \cdot \varphi}}{\varphi} \cdot \frac{(\varphi - 1)^{x \cdot \varphi}}{\varphi}$$

(45)

where $\varphi$ is the number of all possible branch types, and $D = \lvert \mathcal{L}_k \rvert + \lvert \mathcal{E}_k \rvert - 1$

$$D = \lvert \mathcal{L}_k \rvert + \lvert \mathcal{E}_k \rvert - 1$$

(46)

![Figure 30: Graphs for the Proof of Lemma 3](image)

Proof. From the definition, $R = r$ means there are exactly $r$ branches whose vertices or edges are relabelled when transforming $G_1$ into $G_2$ during the graph edit process. For simplicity, we define these branches as **relabelled branches**. However, the value of $R$ could be larger than the difference between branches in $G_1$ and $G_2$, i.e., $GBD(G_1, G_2)$, because it is possible that a subset of relabelled branches are just a re-ordering of the original ones, which are denoted by $\tilde{B}_G$.

For instance, in the graph edit process of transforming $G_1$ into $G_2$, as shown in Figure 30, we need to relabel $v_1$ by label $B$ and $v_4$ by label $A$, which means that the number of relabelled branches $R = 2$. However, $GBD(G_1, G_2) = 0$ since this graph edit process just swaps the branches rooted at $v_1$ and the one rooted at $v_4$, so $\tilde{B}_G = \{ B(v_1), B(v_4) \}$ in this example. To model the probability of this situation, we define $t = \tau - \varphi$, so $t$ is essentially the size of $\tilde{B}_G$. Here we assume that the occurrence probability of each branch type is equal, and denote the number of branch types by $\varphi$. Therefore, $\Omega_3(r, \varphi)$ can be abstracted as the **ball-pair colouring** problem as follows.

- Given two lists of balls $A_1, A_2$ of size $r$, where each ball has been randomly coloured by one of the $\varphi$ colours. We define the **ball pair** as two balls where one ball from $A_1$ and another from $A_2$, where every ball can only be in one pair. What is the probability that there are exactly $t$ ball pairs where the colour inside each pair is the same?

To solve this problem, we take the following steps to count the possible ways to colour balls with the conditions above.

1. The number of ways to form ball pairs is $r!$.

2. When pairs are fixed, the number of ways to select $t$ pairs and assign $t$ colours to them is $\binom{r}{t} \cdot \varphi^t$.

3. For the remaining $r - t$ pairs, the number of ways to assign them different colours inside each pair is $\binom{\varphi - 1}{t - 1}^{r - t}$.

Since there are totally $r! \cdot \varphi^t$ ways to form ball pairs and assign colours to all balls, we have:

$$\Omega_3(r, \varphi) = \frac{r! \cdot \varphi^t}{\varphi^{r - t}} \cdot \binom{\varphi - 1}{t - 1}^{r - t}$$

where the number of possible branch types $\varphi$ is the number of ways to assign $\lvert \mathcal{L}_k \rvert + 1$ labels (including the virtual label) to the vertex in a branch, multiplying the number of ways to assign $\lvert \mathcal{E}_k \rvert$ labels to $\lvert V_k \rvert - 1$ edges in the same branch. This is a variant of the classic object colouring problem and we omit the detailed proof here. □

E. LEMMA 4 AND ITS PROOF

LEMMA 4. Given $GBD = \varphi$, for any integer $r \in [0, \lvert V_1' \rvert]$, we have:

$$\Omega_4(x, r, m) = Pr\{ R = r \mid X = x, Z = m \} = H(x + m - r; \lvert V_1' \rvert, m, x)$$

(47)

where $H(x; M, K, N)$ is defined in Equation (12).

Proof. Let $t = x + m - r$, so $t$ is the number of vertices both relabelled and covered by relabelled edges. Since the order of relabelling operations does not affect the graph edit result, $\Omega_4(x, r, m)$ can be modelled by the following problem:

- First randomly select $m$ vertices from $V'_1$ and tag these vertices as special ones. Then randomly select $x$ vertices from $V'_1$. What is the probability of exactly selecting $t$ special vertices in the second selection?

Since the number of ways to exactly select $t$ special vertices in the second pick is $\binom{\lvert V'_1 \rvert}{m} \binom{\lvert V'_1 \rvert - t}{x - t}$, and the number of all ways to pick $m$ and $x$ vertices separately from $V'_1$ is $\binom{\lvert V'_1 \rvert}{m} \binom{\lvert V'_1 \rvert}{x}$, we have:

$$\Omega_4(x, r, m) = \frac{\binom{\lvert V'_1 \rvert}{m} \binom{\lvert V'_1 \rvert - m}{x - t}}{\binom{\lvert V'_1 \rvert}{m} \binom{\lvert V'_1 \rvert}{x}} = \frac{(m+t)\binom{\lvert V'_1 \rvert - m}{x - t}}{\binom{\lvert V'_1 \rvert}{x}}$$

(47)

where function $H(x; M, K, N)$ is defined in Equation (12). □
F. GRAPH GENERATING ALGORITHM

The algorithm for generating the synthetic graphs (i.e., Syn-1 and Syn-2 data sets) is as follows.

The algorithm aims to generate a set of graphs \( G \), such that for every \( g_i, g_j \in G \), the edit distance between \( g_i \) and \( g_j \) is known. In order to achieve this goal, we first defined a valid modification center, then we designed the generation algorithm which consists two phases: (1) Generate a random “qualified” graph as a template; (2) Modify the template to generate the graph set \( G \).

A modification center is a vertex \( v_c \) in graph \( g \) such that for every \( v_i, v_j \in \{ \text{neighbors of } v_c \}; i \neq j \), the edit distance between \( g - e(i, c) \) and \( g - e(j, c) \) is greater than 0, where \( e(i, c) \) is the edge between vertices \( v_i \) and \( v_c \), and \( e(j, c) \) is the edge between vertices \( v_j \) and \( v_c \). For any modification center \( v_c \), if we randomly modify its adjacent edges, the edit distance between the original and modified graphs can be simply calculated by comparing the edges adjacent to their modification centers in polynomial time.

However, identifying whether a vertex is a modification center is difficult. Therefore, we propose a relatively efficient signature that can help us to filter out some special cases that a vertex is certainly a modification center. For vertex \( v_c \)'s neighbor \( v_i \), the signature is defined as \( \{ s_0, s_1, s_2, \ldots, s_n \} \), where \( s_0 \) is a set contains \( v_i \)'s label, and other sets \( s_k = \{ (v_j, \text{label}, e(i, j).\text{label}); \forall v_j \in k\text{-hop neighbors of } v_j \}; k > 0 \). If a vertex is a modification center, then its neighbors’ signatures must be pair-wised different. That is, if we find a vertex whose neighbors’ signatures are pair-wised different, this vertex must be a modification center. If there is no such a vertex, we re-generate the graph until success.

In our settings, the graph should be connected, and have at least one modification center of degree at least \( d \), to produce a set of graphs among which the edit distance varies from 0 to \( d \). To ensure the connectivity of the graph, we force each vertex \( v_i \) is connected to another vertex \( v_j \), where \( i > j \). After all vertices being connected, we then add remaining edges according to the type of the graph. For random graphs, we randomly add edges between in-adjacent vertices. For scale-free graphs, we add constant number of edges to each vertex \( v_i \), where the neighbors of \( v_i \) is picked from \( \{ v_j; \forall j < i \} \) with the probability proportional to their degrees.