THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, Singapore 117590

TRA6/08

Edit Distance Evaluation on Graph Structures

Zhiping Zeng, Anthony K.H. Tung, Jianyong Wang, Jianhua Feng and Lizhu Zhou

June 2008

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

Edit Distance Evaluation on Graph Structures

Zhiping Zeng[§], Anthony K.H. Tung[†], Jianyong Wang[§], Jianhua Feng[§], Lizhu Zhou[§]
Tsinghua University, Beijing, 100084, P.R.China[§]
National University of Singapore, Singapore[†]
cengzp03@mails.tsinghua.edu.cn, atung@comp.nus.edu.sg
{jianyong, fengjh, dcszlz}@tsinghua.edu.cn

ABSTRACT

Graph data has became ubiquitous and manipulating them based on similarity is essential for many applications. Graph edit distance is one of the most widely accepted measure to determine similarities between graphs and has extensive applications in the fields of pattern recognition, computer vision etc. Unfortunately, the problem of graph edit distance computation is NP-Hard in general and is unlikely to be approximated within a polynomial time computable factor $f(n)$ in polynomial time. Accordingly, in this paper we introduce three novel methods to compute the upper and lower bounds for the edit distance between two graphs in polynomial time. Applying these methods, three algorithms AppFull, ExactSub and AppSub are introduced to perform different kinds of graph search on graph databases. Comprehensive experimental studies are conducted on both real and synthetic datasets to examine various aspects of the methods for bounding graph edit distance. Result shows that these methods achieve good scalability in terms of both the number of graphs and the size of graphs. The effectiveness of these algorithms also confirms the usefulness of using our bounds in filtering and searching of graphs.

1. INTRODUCTION

In the modern society, graph data are becoming ubiquitous, and graph data models have been studied in the database community for semantic data modelling, hypertext, multimedia, chemical and biological information system. For example, World Wide Web can be considered as a graph whose vertices correspond to static pages and edges correspond to links between pages[6]; in chem-informatics, labelled graphs are suited to express the connectivity of chemical compounds[29]; in bioinformatics, collections of DNA segments in a cell which interact with each other and with other substances in the cell can be formatted as gene regulatory networks[4, 5, 15].

Due to the extensive applications of graph models, vast amounts of graph data have been collected and graph databases have attracted significant attention in the academic communities. In recent years, various approach have been proposed to deal with a variety of graph-related research problems. For example, a variety of effective algorithms have been devised to mine graph patterns(e.g., frequent patterns) from graph databases[33], to index graph databases for efficiently processing graph search[13, 37, 8] and to perform keyword search over graph databases[14, 17, 30].

With the rapidly increasing amounts of graph data(e.g., chemical compounds and social network data), supporting scalable graph search over large graph databases becomes an important database research problem. Because of the technical limitation of processing graph search using conventional database technologies, enormous efforts [26, 13, 35, 8, 37] have been put into constructing practical graph searching methods.

Given a graph database consisting of n graphs, $\mathcal{D} = \{g_1, g_2, \cdots, g_n\}$, and a query graph q, almost all existing algorithms of processing graph search can be classified into the following three categories:

1. ***full graph search***: find all graphs g_i in $\mathcal{D}$ s.t. g_i is the same as q [3];
2. ***subgraph search***: find all graphs g_i in $\mathcal{D}$ containing q [4, 26, 27, 37] or contained by q[8].
3. ***similarity search***: find all graphs g_i in $\mathcal{D}$ s.t. g_i is similar to q within a user-specified threshold based on some similarity measures [23, 31].

As can be seen, manipulating graph data based on structural similarity is essential for many applications [36]. A number of graph similarity measures therefore have been proposed [20, 7, 10, 24]. Among these, **graph edit distance** has been widely accepted as a similarity measure for representing the distances between attributed graphs. Informally speaking, graph edit distance defines the similarity of two graphs by the minimum amount of distortion which is needed to transform one graph into the other. In contrast with other measures, graph edit distance does not suffer from any restrictions and can be applied to any type of graphs. Furthermore, graph edit distance is known to be error-tolerant with the ability to find graphs that are of interest to users even in the presence of noises and errors in the database.

Graph edit distance plays a significant role in the management of graph data and a variety of other applications such

as graph classification, computer vision, pattern recognition and etc. For example, a familiar problem in computer vision is to recognizing specific objects within an image[16](e.g., face identification and symbol recognition). In this case, a representative graph is generated from the image according to structural characteristics, and vertex labels may be assigned based on characteristics of the region to which each vertex corresponds. After then, this representative graph is compared to a database of prototype or model graphs to identify and classify the object of interest. In this context, graph edit distance provides a good measure for comparing graphs.

Unfortunately, the main drawback of graph edit distance is its exponential computation complexity in terms of the number of graph vertices. As will be shown in Section 2, the problem of graph edit distance is NP-hard in general, and it is even unlikely to have an algorithm that can approximate it within a factor $f(n)$ in polynomial time, where $f(n)$ is any polynomial time computable function. The direct computation of graph edit distance involving large graphs is therefore expensive and will take an unacceptable time. Because of this, a few algorithms have been proposed to compute upper and lower bounds for the graph edit distance, each having their own disadvantages. Justice et al. [16] gave a solution providing the lower bound in $O(n^7)$ time and the upper bound in $O(n^3)$ time. The computation of the lower bound is expensive, and their method for obtaining the upper bound will consider only vertex edit term, i.e., without considering structural information in graphs. This solution is therefore not applicable in practice. The other methods [22, 32] cannot provide lower bounds, and used heuristic algorithms to find unbounded suboptimal values. All their computation complexities are hard to analyze and not presented in related papers.

Accordingly, in this paper we address the problem of obtaining upper and lower bounds of graph edit distance efficiently. In summary, the contributions of this paper are:

- We give a formal proof that the problem of graph edit distance computation is NP-Hard and it cannot be approximated within a factor of $f(n)$ unless $P = SPP$, where $f(n)$ is a polynomial time computable function.
- We introduce a notion of *star representations* for graph structures and propose three novel methods to obtain lower and upper bounds of edit distance between two graphs in polynomial time.
- Based on these efficiently computable bounds, we developed three algorithms AppFull, ExactSub and AppSub for performing approximate full graph search, exact subgraph search and approximate subgraph search respectively.
- Comprehensive experimental studies are conducted to evaluate the scalability and effectiveness of our algorithms.

The rest of this paper is organized as follows. Section 2 will formalize the problem of graph edit distance computation together with computation complexity analysis on the problem. Related work will be discussed in Section 3. In Section 4, three efficiently computable methods are introduced for obtaining lower and upper bounds of graph edit distance. Section 5 investigates the applications of these bounds in performing graph search over graph databases, followed by a comprehensive experimental studies reported in Section 6. Section 7 concludes the paper.

2. PRELIMINARIES

In this section, we will first formalize the problem of graph edit distance computation and then perform some computational complexity analysis for the problem. Table 1 summarizes the major notations that we will use in this paper.

Symbols	Description
$deg(v)$	$\lvert\{u\lvert(u,v) \in E\}\rvert$, the degree of v
$\delta(g)$	$\max_{v \in V(g)} deg(v)$
$\lambda(g_1, g_2)$	the edit distance between graphs g_1 and g_2
$L_m(g_1, g_2)$	the lower bound of $\lambda(g_1, g_2)$
$\tau(g_1, g_2)$	the suboptimal value of $\lambda(g_1, g_2)$
$\rho(g_1, g_2)$	the refined suboptimal value $\lambda(g_1, g_2)$

Table 1: Notations used in this paper

2.1 Problem Formulation

In this paper, we consider ***simple graphs*** which does not contain self-loops, multi-edges and edge labels. An ***undirected attributed graph***, denoted by g, can be represented by a 3-tuple $g=(V, E, l)$, where V is a finite set of vertices, $E \subseteq V \times V$ is a set of vertex pairs, and $l : V \rightarrow \Sigma$ [1] is a function assigning labels to vertices. In general, we will use E_i, V_i and l_i to represent the edges, nodes and label assigning function of a graph g_i. Without explicit statement, the term ***graph*** we refer to in the rest of this paper means an undirected attributed graph.

A graph g_1 is ***subgraph isomorphic*** to another graph g_2(denoted by $g_1 \sqsubseteq g_2$) iff there exists an injection $f : V_1 \rightarrow V_2$ such that for any vertex $v \in V_1$, $f(v) \in V_2 \wedge l_1(v) = l_2(f(v))$, and for any edge $(u, v) \in E_1$, $(f(u), f(v)) \in E_2$. Moreover, if g_1 and g_2 are subgraph isomorphic to each other, they are said to be ***graph isomorphic*** (denoted by $g_1 = g_2$).

Informally speaking, an ***edit operation*** on a graph g is an insertion or deletion of a vertex/edge or relabelling of a vertex. A vertex can be deleted only on the condition that no edge is connected to the vertex, and the costs of different edit operations are assumed to be equal in this paper. Essentially, a vertex deletion can be considered as a vertex relabelling by changing its label from $\sigma \in \Sigma$ to ϵ where ϵ is a special label indicating that the vertex is virtual. Symmetrically, a vertex insertion can be considered as relabelling a vertex's label from ϵ to σ. Let p_i denote an edit operation, an ***alignment*** P is an edit operation sequence $\langle p_1, p_2, \ldots, p_m \rangle$ which can transform a graph g_1 into another graph g'. If g' is graph isomorphic to another graph g_2, we say g_1 can ***reach*** g_2 through P. Alignments which can make g_1 reach g_2 are not unique, and ***optimal alignments*** between g_1 and g_2 are alignments containing a minimum number of edit operations.

Definition 2.1. **(Graph Edit Distance)** *The edit distance between g_1 and g_2, denoted by $\lambda(g_1, g_2)$, is the number of edit operations in the optimal alignments that make g_1 reach g_2.*

[1] Σ is an alphabet consisting of all labels.

2.2 Computation Complexity Analysis

In the above, the formulation and properties of the GED problem have been introduced. Next, we investigate its computational complexity. Justice et al.[16] used the adjacency matrix representation to formulate a BLP(binary linear program, a linear program where all variables are either 0 or 1) to solve the GED problem. Because solving a BLP is NP-Hard [11] in general, it is likely that GED problem is NP-Hard. We confirm this possibility here.

LEMMA 2.1. *Given two graphs g_1 and g_2, $\lambda(g_1, g_2) \geq ||E_2| - |E_1|| + ||V_2| - |V_1||$.*

PROOF. Given that the edit operations had transformed g_1 to g_2, then the transformed graph of g_1 must have the same number of nodes and edges as g_2. It should be easy to see that at least $||E_2| - |E_1|| + ||V_2| - |V_1||$ edit operations are needed to do so. □

LEMMA 2.2. *Given two graphs g_1 and g_2 where $|V_1| \leq |V_2|$ and $|E_1| \leq |E_2|$, g_1 is subgraph isomorphic to g_2 iff $\lambda(g_1, g_2) = (|E_2| - |E_1|) + (|V_2| - |V_1|)$.*

PROOF. ***Necessity:*** if g_1 is subgraph isomorphic to g_2, then there exists an injection η that satisfy the required condition. We define $\mathbb{V} = \{u \,|\, \exists\, v \in V_1, \eta(v) = u\}$ and $\mathbb{E} = \{(\eta(u), \eta(v)) | (u, v) \in E_1\}$. Since η is injective, $\mathbb{V} \subseteq V_2$, $\mathbb{E} \subseteq E_2$, $|\mathbb{E}| = |E_1|$ and $|\mathbb{V}| = |V_1|$ must hold. Suppose P is an alignment which removes all edges in $E_2 - \mathbb{E}$ and all vertices in $V_2 - \mathbb{V}$, then g_2 can reach g_1 through P. Based on Lemma 2.1, P is an optimal alignment which makes g_2 reach g_1, thus $\lambda(g_1, g_2) = (|E_2| - |E_1|) + (|V_2| - |V_1|)$.

Sufficiency: assume P is an alignment that makes g_2 reach g_1. In order to converge g_2 to g_1 by performing P on g_2 possess the identical number of edges and vertices, at least $(|E_2| - |E_1|) + (|V_2| - |V_1|)$ delete operations exist in P. Thus, if P is an optimal alignment containing $(|E_2| - |E_1|) + (|V_2| - |V_1|)$ edit operations, neither insertion nor vertex relabelling exists in P. The graph isomorphism from g_1 to g' is therefore a subgraph isomorphism from g_1 to g_2, i.e., $g_1 \sqsubseteq g_2$. □

Therefore, graph edit distance can also be used to determine the subgraph isomorphism which is NP-Complete [11]. Then we can derive the following lemma.

LEMMA 2.3. *GED problem is NP-Hard.*

PROOF. For two graphs g_1 and g_2, if $|V_1| > |V_2|$ or $|E_1| > |E_2|$, we can quickly state that g_1 cannot be subgraph isomorphic to g_2 because it is impossible to find such a subgraph isomorphism. In the case of $|V_1| \leq |V_2|$ and $|E_1| \leq |E_2|$, based on Lemma 2.2, the subgraph isomorphism between g_1 and g_2 which is NP-Complete [11] can be reduced to GED problem in polynomial time. GED problem is therefore NP-Hard. □

According to Lemma 2.3, it is prohibitively difficult to compute the graph edit distances for large graphs. In this situation, people usually look for approximation algorithms to avoid the expensive computation. Unfortunately, according to Lemma 2.4 introduced below, it is unlikely to approximate GED problem within a polynomial time computable factor $f(n)$ unless P=SPP, where SPP is a complexity class first introduced in [9]. To date, P is contained in SPP, but SPP is not known to be contained in NP. Also, SPP is not known to contain any NP-complete problems, but SPP does contain some problems believed to be hard, e.g., it contains decision versions of discrete logarithm and integer factoring. Therefore, it is unlikely that SPP is in P.

LEMMA 2.4. *For any polynomial time computable function $f(n)$, the graph edit distance between two graphs cannot be approximated within a factor of $f(n)$, unless P=SPP.*

PROOF. Assume on the contrary that there is a factor $f(n)$ polynomial time approximation algorithm, $\mathcal{A}$, for general GED problem. We will show that $\mathcal{A}$ can be used for deciding the graph isomorphism(GI) problem which is SPP[2] in polynomial time, thus implying P=SPP.

The central idea is a reduction from the GI problem to GED problem. Given two graphs g_0 and g_1 whose vertex labels are identical, then g_0 and g_1 can be treated as general graphs without vertex label. If g_0 is graph isomorphic to g_1, then $\lambda(g_0, g_1) = 0$ must hold. Observe that when running on g_0 and g_1, algorithm $\mathcal{A}$ must return a result of $\mathcal{A}(g_0, g_1) \leq f(n) \times \lambda(g_0, g_1) = 0$ if g_0 is isomorphic to g_1. Otherwise, $\mathcal{A}$ must return a value of greater than 0. Therefore, $\mathcal{A}$ can be used to determine the graph isomorphism between g_0 and g_1. □

Based on these analyses, we conclude that heuristic approaches to approximate GED will most probably be the best way to go towards solving the problem of computing graph edit distance.

3. RELATED WORK

3.1 Graph Edit Distance

There are a number of existing studies addressing the graph edit distance computation problem [22, 25]. All of them fall into two categories: ***exact algorithms*** and ***heuristic algorithms***.

The most widely used method for computing exact graph edit distance is based on the well-known ***A* algorithm***[12], and Kaspar Riesen et al. used bipartite heuristic to speed up the computation procedure[25]. However, as stated in [22], in practice this kind of algorithms are practical for computing the edit distance of graphs typically possessing 12 vertices or less. Exact algorithms therefore cannot be applied in the applications involving large graphs, and plenty of heuristic algorithms are devised with to compute lower bound and upper bound for GED with unbounded errors.

Exploiting the strategy of the A* algorithm, Michel Neuhaus et al. proposed a heuristic algorithm [22] by maintaining only a fixed number of nodes with the lowest cost and introducing an additional weighting factor favoring long partial edit paths over shorter ones. In the community of pattern recognition, the GED problem is named as graph matching. From the standpoint of information theory, it is seeking the matched configuration of vertices that has maximum a posteriori probability w.r.t. the available vertex attribute information. As, graph matching algorithms aims to optimize a global MAP criterion[32, 21], some heuristic algorithms are devised based on this framework[21, 32]. However, it is hard to analyze the computation complexities of the above heuristic algorithms, and the suboptimal solutions provided by them are also unbounded.

Meanwhile, the authors in [1] and [16] formulated the GED problem as a BLP problem[2]. As this formulation will be used in Section 4.3, it is therefore described in detail.

[2]Although the work in [1] focused on weighted graph matching problem, it can be considered as a special case of graph edit distance on graphs without vertex labels.

The adjacency matrix A^g for g is given by $A^g=\{a_{i,j}\}$, where

$$a_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E(g) \\ 0, & \text{otherwise} \end{cases}$$

For two graphs g and h, assume $|V(g)|=n$ and $|V(g)|\geq|V(h)|$, then vertices with the special label ϵ are inserted into h such that h will contain n vertices. Let $P=\{P_{i,j}\}$ be an $n\times n$ permutation matrix and $C=\{C_{i,j}\}$ be an $n\times n$ label matrix where

$$P_{i,j} \in \{0,1\}, \qquad \sum_{i=1}^{n} P_{i,j} = \sum_{j=1}^{n} P_{i,j} = 1$$

$$C_{i,j} = \begin{cases} 1, & \text{if } l_g(v_i) = l_h(u_j), v_i \in V(g), u_j \in V(h) \\ 0, & \text{otherwise.} \end{cases}$$

Next, let P be an orthogonal matrix having the property $PP^T=P^TP=I$ where I is the identity matrix. For each permutation matrix P, the cost of transforming g to h using P, $C(g,h,P)$, is defined as

$$C(g,h,P) = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{i,j}P_{i,j} + \frac{1}{2}||A^g - PA^hP^T||_1 \qquad (1)$$

where $||.||_1$ denotes the L_1 norm, i.e.,

$$||A||_1 = \sum_{i=1}^{n}\sum_{j=1}^{n} |a_{i,j}|$$

The edit distance between g and h can be formulated as

$$\lambda(g,h) = \min_P C(g,h,P) \qquad (2)$$

It is shown that the problem of graph edit distance is equivalent to find an optimal permutation matrix P^* to minimize $C(g,h,P)$. According to Lemma 2.3, it is prohibitively difficult to find P^* directly. Hence, some techniques are proposed to overcome this restriction. Let

$$R = A^g - PA^hP^T \qquad (3)$$

Since P is a permutation matrix, the L_1 norm of (3) is

$$||R||_1 = ||RP||_1 = ||A^gP - PA^h||_1 \qquad (4)$$

Let $VEC(A)$ be the column vector obtain by stacking the columns of A on top of one another. Then, (4) can be written in the form

$$VEC(RP) = A^{gh} \cdot VEC(P)$$

where A^{gh} is an $n^2 \times n^2$ constant matrix derived from g and h. It is clear from the above transformation that $||A^gP - PA^h||_1 = ||A^{gh}VEC(P)||_1$. Then, (2) can be written as:

$$\lambda(g,h) = \min_P ||A^{gh}VEC(P)||_1 + \sum_{i=1}^{n}\sum_{j=1}^{n} C_{i,j}P_{i,j} \qquad (5)$$

A consequent advantage of (5) is that (2) is transformed from a nonlinear optimization problem to a linear optimization problem, and a lower bound of λ can be obtained in $O(n^7)$ time by extending the domain of P from $\{0,1\}$ to $[0,1]$. Moreover, an upper bound of λ also can be obtained in $O(n^3)$ time with only vertex edit term, i.e., connectivity information are not considered during the computation of obtaining the upper bound. Accordingly, this upper bound will not be tight. More details about this solution can be found in [16].

3.2 Graph Search

Meanwhile, there are a wealth of literature concerning the problem of graph search, and a large number of algorithms have been devised. Due to the complexity of (sub)graph isomorphism, people usually exploit the feature-based indexing approach in practical graph search systems to improve the performance. In content-based image retrieval, Petrakis et. al represented each graph a vector of features and indexed graphs in multidimensional space using R-trees[23]. Instead of casting graphs to vectors, a metric indexing scheme is proposed which organizes graphs hierarchically by means of their mutual distances[3]. All the above systems are designed to address the full graph search problem.

For subgraph search systems, almost all of them exploit ***path-based*** or ***graph-based*** indexing approach. **GraphGrep**[26] is a famous representative of path-based approach. Srinivasa et al. built multiple abstract graphs[27], **gIndex**[34] uses the discriminative frequent structures, **C-Tree**[13] adopted graph closure, **SAGA**[28] employed pathway fragments, **cIndex**[8] adopted contrast subgraphs while **Tree+Δ** [37] exploited frequent tree-features(Tree) and a small number of discriminative graphs(Δ).

Because of noises that are usually present in graph databases, a common problem in graph search is: "what if there is no match or very few matches for a given query?"[35]. In this scenario, similarity search becomes an appealing and natural choice. There are also a number of systems supporting similarity search over graph databases. For example, a three-tier algorithm for full graph similarity search is introduced by Raymod et al.[24] whose similarity is based on *maximum common edge subgraph*; He et al. in [13] exploited approximate graph edit distance computed through heuristic neighbor biased mapping methods; the similarity measure proposed in [28] consists of three components, *StructDist, NodeMismatches and NodeGaps*; while *Grafil*[35] supports approximate subgraph search by allowing edge relaxations.

4. GRAPH EDIT DISTANCE EVALUATION

In this section, we present methods to efficiently obtaining the lower and upper bounds of edit distance between two graphs.

4.1 Graph Transformation

The key idea of this paper is to transform a graph structure to a multiset of star structures. This transformation retains certain structural information of the original graph structure.

4.1.1 Star Structure

DEFINITION 4.1. **(Star Structure)** *A star structure s is an attributed, single-level, rooted tree which can be represented by a 3-tuple $s=(r,L,l)$, where r is the root vertex, L is the set of leaves and l is a labeling function. Edges exist between r and any vertex in L and no edge exists among vertices in L.*

In a star structure, the root vertex is the center and vertices in L can be considered as satellites surrounding the cen-

ter. For any vertex v_i in a graph g, we can generate a corresponding star structure s_i in the following way: s_i=(v_i, L_i, l) where L_i=$\{u|(u, v_i) \in E\}$. Accordingly, we can derive n star structures from a graph containing n vertices. Thus, a graph can be mapped to a multiset[3] of star structures. We call this multiset the ***star representation*** of the graph g, denoted by $S(g)$.

4.1.2 Star Edit Distance

Due to the particularity of star structures, edit distance between two star structures can be computed easily as shown below.

LEMMA 4.1. **(Star Edit Distance)** *Given two star structures s_1 and s_2,*

$$\lambda(s_1, s_2) = T(r_1, r_2) + d(L_1, L_2)$$

where

$$T(r_1, r_2) = \begin{cases} 0 & \text{if } l(r_1) = l(r_2), \\ 1 & \text{otherwise.} \end{cases}$$

$$d(L_1, L_2) = ||L_1| - |L_2|| + M(L_1, L_2)$$

$$M(L_1, L_2) = \max\{|\Psi_{L_1}|, |\Psi_{L_2}|\} - |\Psi_{L_1} \cap \Psi_{L_2}|$$

Ψ_L is the multiset of vertex labels in L.

PROOF. This proof is simple and we omit it here. □

Based on the above lemma, the main cost for computing edit distance between two star structures is the multiset intersection operation. To speed up this operation, a total order should be defined on Σ. For instance, we can attach distinct integers to distinct vertex labels. After that, a multiset can be sorted in ascending order with computation complexity $O(n \log n)$. Note that, this sort operation can be accomplished during the preprocessing of input databases. ALGORITHM 1 illustrates an efficient method to compute the intersection of two multisets in which elements are sorted in ascending order. The analysis of this algorithm is quite simple, and its computation complexity is $\Theta(n)$.

Algorithm 1 MI - Multiset Intersection

Input: two sorted vectors A and B
Output: C - the elements appear in both A and B

```
1.  i ← 0, j ← 0;
2.  while i < |A| and j < |B| do
3.    if A[i] = B[j] then
4.      C ← C ∪ A[i];
5.      i++,j++;
6.    else if A[i] < B[j] then
7.      i++;
8.    else if A[i] > B[j] then
9.      j++;
10.   end if
11. end while
12. return C;
```

4.2 Lower Bound of Edit Distance

Based on the star representation of graphs, a polynomial time computable distance L_m is introduced in this subsection to give a lower bound for graph edit distance.

[3]NOT "set" here, since multiple star structures could appear.

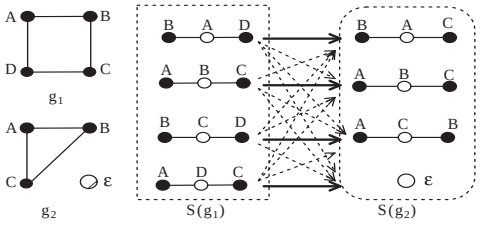


Figure 1: Computing $\zeta(S_1, S_2)$: A Bipartite Graph Matching Problem

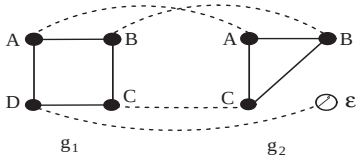


Figure 2: Mapping between Nodes of Two Graphs in an Optimal Alignment

4.2.1 Mapping Distance

We first define the distance between two multisets of star structures. Subsequently, we will define the mapping distance between two graphs based on their star representations which are multisets.

DEFINITION 4.2. *Given two multisets of star structures S_1 and S_2 with the same cardinality, and assume P:$S_1 \to S_2$ is a bijection. The distance ζ between S_1 and S_2 is*

$$\zeta(S_1, S_2) = \min_P \sum_{s_i \in S_1} \lambda(s_i, P(s_i))$$

The computation of $\zeta(S_1, S_2)$ is equivalent to solving the assignment problem, which is one of the fundamental combinational optimization problem aims at finding the minimum weight matching in a weighted bipartite graph. Given two set of vertices V and V' in a weighted graph, the problem aim to find a set of edges E such that each vertex in V is link to exactly one vertex in V' such that the sum of weight for edges in $|E|$ is minimized. In our case, these two set of vertices are the two multisets of star structures S_1 and S_2 and the weight of the edge that connect one star in S_1 to another star in S_2 is the edit distance between the two stars. Figure 1 show an example to illustrate the bipartite graph matching problem that must be solved in order to compute $\zeta(S(g_1), S(g_2))$. Given two graph g_1 and g_2 on the left of the figure, their corresponding multisets of star structures are shown on the right hand side of the figure. In this case, we show the optimal matching between the stars as solid line while other edges joining the stars are shown as dotted line.

Note that in Figure 1, the two graphs have different number of nodes and hence an additional node labelled as ϵ was added in S_2. We will now address this issue. Assume $|V_1|-|V_2|=k \geq 0$, $|S(g_1)|-|S(g_2)|=k$ must hold. In order to make them have the same number of vertices, k vertices with the label ϵ are inserted into g_2. We call this process the normalization of g_2 w.r.t. g_1. Because vertex insertion can be

considered as vertex relabelling from ϵ to σ, this normalization will not change the edit distance between g_1 and g_2. After this normalization, $S(g_1)$ and $S(g_2)$ possess the same cardinality, and the mapping distance between two graphs can be defined upon the distance of their star representations.

In order to solve the bipartite graph matching problem, we first create an $n \times n$ matrix in which each element represents the edit distance between the ith star in $S(g_1)$ and the jth star in $S(g_2)$. The ***Hungarian algorithm***[18] is then applied on this square matrix to obtain the minimum cost in $O(n^3)$ time. We now formally definite the ***mapping distance*** between two graphs.

DEFINITION 4.3. **(Mapping Distance)** *The mapping distance $\mu(g_1, g_2)$ between g_1 and g_2 is defined as:*

$$\mu(g_1, g_2) = \zeta(S(g_1), S(g_2))$$

Intuitively, the optimal mapping for computing the mapping distance between $S(g_1)$ and $S(g_2)$ is trying to approximate the mapping between the nodes of g_1 and g_2 in an optimal alignment. Figure 2 shows the mapping between the nodes of the two graphs that are shown in Figure 1 when they are optimally aligned. In this case, the optimal mapping computed on the bipartite graph is in fact the same as the mapping in the optimal alignment. Note that since the mapping in the bipartite graph take only each nodes and their neighbors into consideration, there is in fact less constraints on the output when determining the optimal mapping in the bipartite graphs compare to determining the optimal mapping between the two graphs. Because of this, it is possible to compute a lower bound of the edit distance for g_1 and g_2 by utilizing $\mu(g_1, g_2)$. We will prove this lower bound formally in the next section.

Furthermore, since the mapping computed on the bipartite graph might not be the same as the mapping computed for the optimal alignment between g_1 and g_2, the number of edit operation that convert g_1 into g_2 based on the bipartite graph mapping[4] will either be the same or higher than the actual edit distance between the two graphs. Later on in this paper, we will utilize this observation to compute an upper bound on the edit distance between two graphs.

4.2.2 Lower Bound of Graph Edit Distance

LEMMA 4.2. *Let g_1 and g_2 be two graphs, then the mapping distance $\mu(g_1, g_2)$ between g_1 and g_2 satisfies the following:*

$$\mu(g_1, g_2) \leq \max\{4, [\max\{\delta(g_1), \delta(g_2)]\} + 1]\} \cdot \lambda(g_1, g_2)$$

PROOF. Let P=$(p_1, p_2, \ldots, p_k)$ be an alignment transforming g_1 to g_2. Accordingly, there is a sequence of graphs g_1=$h_0 \rightarrow h_1 \rightarrow \ldots \rightarrow h_k$=$g_2$, where $h_{i-1} \rightarrow h_i$ indicates that h_i is the derived graph by performing p_i over h_{i-1} for $1 \leq i \leq k$. Assume there are k_1 edge insertion/deletion operations, k_2 vertex insertion/deletion operations and k_3 vertex relabellings in P, then k_1+k_2+k_3=k. Next, we will analyze each kind of edit operations in detail.

1. ***Edge Insertion/Deletion***: while an edge is inserted between two vertices v_i and v_j in the graph h_m, only two star structures rooted by v_i and v_j are affected. For the star rooted by v_i, a new vertex v_j and a new edge (v_i, v_j) are newly inserted into this star structure after the edge insertion, as illustrated in Figure 3. Likewise, the same effect is incurred by the star structure rooted at v_j. Thus, we can know that $\mu(h_m, h_{m+1}) \leq 2 \times 2 = 4$. In complement, one edge deletion has the same affection. Therefore, in the case of performing one edit operation of inserting or deleting a edge on h_m, $\mu(h_m, h_{m+1}) \leq 4$.

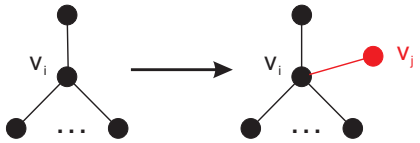


Figure 3: Star changes incurred by edge insertion

2. ***Vertex Insertion/Deletion***: In the case of vertex insertion, since the newly inserted vertex v_0 has no edge attached, it is equivalent to changing a vertex's label from ϵ to σ. Thus, in the case of inserting one vertex, $\mu(h_m, h_{m+1}) = 1$. The same result can be induced in case of deleting one vertex due to the complementary.
3. ***Vertex Relabeling***: Assume a vertex v_0's label is changed from σ_1 to σ_2. Obviously, the star rooted by v_0 and $deg(v_0)$ stars which possess v_0 as their leaves are affected by this relabelling. For each such star, only the label of v_0 is changed. Therefore, for one vertex relabelling operation, $\mu(h_m, h_{m+1}) \leq 1 \times (deg(v_0)+1)$.

Above all, we will get the following inequality

$$\begin{aligned}\mu(g_1, g_2) &\leq 4 \cdot k_1 + 1 \cdot k_2 + (\max\{\delta(g_1), \delta(g_2)\} + 1) \cdot k_3 \\ &\leq \max\{4, [\max\{\delta(g_1), \delta(g_2)\} + 1]\} \cdot (k_1 + k_2 + k_3) \\ &\leq \max\{4, [\max\{\delta(g_1), \delta(g_2)\} + 1]\} \cdot \lambda(g_1, g_2)\end{aligned}$$

This complete the proof. □

Based on Lemma 4.2, μ provides a lower bound L_m of λ, i.e.,

$$\lambda(g_1, g_2) \geq L_m(g_1, g_2) = \frac{\mu(g_1, g_2)}{\max\{4, [\max\{\delta(g_1), \delta(g_2)\} + 1]\}}$$

Before applying Hungarian algorithm, a $n \times n$ matrix must be constructed and the computational complexity for computing edit distances among star structures is $\Theta(n)$, thus the cost of this construction is $\Theta(n^3)$. Since the computational complexity of Hungarian algorithm is $O(n^3)$, both μ and L_m therefore can be computed in $\Theta(n^3)$ time.

4.3 Upper Bound of Edit Distance

In previous subsection, L_m is devised as a lower bound for the graph edit distance λ. Next, we will introduce two algorithms to compute upper bounds for λ in this section.

The first upper bound comes naturally during the computation of μ introduced in Section 4.2.1. As mentioned, since the optimal mapping that is computed for the bipartite graph in Section 4.2.1 are done by considering only each vertex and it's neighbors, there is in fact less constraints compared to when computing the edit distance between their corresponding graphs. Assuming that the output from the Hungarian algorithm in Section 4.2.1 leads to

[4]This can be computed using Equation 1 in Section 3.1.

a mapping $\bar{P}$ from $V(g)$ to $V(h)$, we can simply use Equation 1 from Section 3.1 to compute $C(g,h,\bar{P})$, denoted as $\tau(g,h)$. Apparently, since the mapping might not be optimal $\tau(g,h) \geq \lambda(g,h)$, the actual edit distance between g and h. Because $C(g,h,P)$ can be solved in $\Theta(n^2)$ time, $\tau(g,h)$ is therefore an upper bound of $\lambda(g,h)$ that can be computed in $\Theta(n^3)$ time.

While $\tau(g,h)$ give an initial upper bound on the edit distance, it is possible to perform an iterative refinement approach on the bipartite graph mapping in order to improve the approximate upper bound. The main idea is that given any two nodes u_1 and u_2 in g and their corresponding mapping $f(u_1)$ and $f(u_2)$ in h (assuming f is the mapping function corresponding to $\bar{P}$), we swap $f(u_1)$ and $f(u_2)$ if this reduce the edit distance. As such, for any pair of $(u_1,u_2) \in V(g)$, a new mapping function f' can be defined as following:

$$f'(u) = \begin{cases} f(u) & \text{if } u \neq u_1 \text{ and } u \neq u_2 \\ f(u_2) & \text{if } u = u_1 \\ f(u_1) & \text{if } u = u_2 \end{cases}$$

Let P' denote the permutation matrix corresponding to f'. Because there are C_n^2 pairs of vertices in $V(g)$, for a mapping function f there will be C_n^2 newly generated permutation matrices. For each P' we can get obtain a new value $C(g,h,P')$. Assume P_0 is the permutation matrix which results in the minimum value of $C(g,h,P')$, i.e.,

$$P_0 = \arg\min_{P'} C(g,h,P')$$

If $C(g,h,P_0) < C(g,h,\bar{P})$, then we get a closer upper bound of $\lambda(g,h)$. After that, we assign $\bar{P}$ to be P_0 and continue performing the refinement on $\bar{P}$ until $C(g,h,P_0) \geq C(g,h,\bar{P})$. Algorithm 2 illustrates this iterative refinement approach in detail.

Algorithm 2 Refine(g,h,P)

Input: two graph structures g and h
Input: a permutation matrix P of g and h
Output: refined suboptimal distance of g and h

```
1.  dist ← C(g, h, P);
2.  min ← dist;
3.  for any pair (u_i, u_j) ∈ V(g) do
4.     get P' based on u_i and u_j;
5.     if min > C(g, h, P') then
6.        min ← C(g, h, P');
7.        P_min ← P';
8.     end if
9.  end for
10. if min < dist then
11.    min ← Refine(g, h, P_min)
12. end if
13. return min;
```

Because the optimization problem shown in Equation (2) is not a linear optimization problem, Refine will only find a local optimal solution. ρ is therefore also an upper bound of λ as well as τ. The relationships between L_m, τ, ρ and λ can thus be represented by the inequality $L_m \leq \lambda \leq \rho \leq \tau$. In addition, because ρ is always no less than λ, Refine is guaranteed to terminated. Because Refine is an iterative algorithm, its computational complexity is difficult to analyze. However, the value of τ will not exceed the total number of vertices and edges residing in these two graphs, i.e., $\tau \leq 2\times(n+0.5n^2) = 2n+n^2$ where n is the vertex number in involved graphs. Refine is therefore guaranteed to be terminated in $2n+n^2$ steps. Moreover, the cost for each step is $C_n^2 \times n^2$, as such the computational complexity Refine is at most $O(n^6)$. Note that, the above run time complexity of ρ is a theoretical extreme case. In practise, it performs better and will converge after small number of iterations.

5. APPLICATIONS IN GRAPH SEARCH

Next, we will look at how the bounds developed in previous section can be utilized to perform various types of graph searching. As described in Section 2, graph edit distance can be used to measure structural similarity and also determine subgraph isomorphism. All three kinds of graph search listed in Section 1 can therefore be done using graph edit distance as a similarity measure. However, the problem of GED computation is NP-Hard, and the exact computation is very expensive. In this case, we exploit the upper and lower bounds of edit distance to improve the search performance by filtering out graphs that definitely will not be in the answer set and thus avoid the expensive graph edit distance computation.

5.1 Approximate Full Graph Search

Because of the existence of noise in graph databases, graph search with approximation is more preferable. To the best of our knowledge, while there are studies dealing with similarity search over graph databases using specific similarity measures, no algorithm has been proposed in which graph edit distance is used as the similarity measure. Here, we will use the three bounds of λ that we have developed earlier to develop an effective algorithm AppFull for performing approximate full graph over graph databases using graph edit distance as the similarity measure.

As shown in Algorithm 3, a multi-level composition strategy based on L_m, τ and ρ is adopted in AppFull. Given a query graph q and the edit distance threshold l, for each graph g in graph database $\mathcal{D}$, $L_m(g,q)$ is first used to filter g if L_m is greater than l(lines 2-4), because $\lambda > l$ must hold in the case of $L_m > l$. Subsequently, if $\tau(g,q) \leq l$, we know that the edit distance between g and q must be no greater than l, and q can therefore be reported as a result(lines 5-8). If g passes the above two tests, then ρ is exploited further. Like in the case of τ, if $\rho(g,q) \leq l$, $\lambda(g,q)$ must be no greater than l and g can be output as a result (lines 9-12). Finally, if g passes all the above three tests, then $\lambda(g,q)$ must be computed(lines 13-15). The order of the above three tests is quite significant because the costs of their computation are different. Among three of them, the computation of L_m is the most efficient, while the computation of ρ is the most expensive. Therefore, if g does not pass an earlier test, the rest of the expensive tests are avoided. This arrangement of tests therefore make AppFull more efficient. In addition, from AppFull, we can see that, the expensive computation of λ is only conducted for graphs that pass all the three tests and a large number of λ computation are therefore avoided. The performance of AppFull will be evaluated in the experimental study section.

5.2 Subgraph Search

We next look at subgraph search [13, 37, 35, 8]. We will first look at exact subgraph search which by itself is NP-complete. Then, we will look at approximate subgraph search.

Algorithm 3 AppFull - Approximate Full Graph Search

Input: A query graph q and a graph database $\mathcal{D}$
Input: Distance threshold l
Output: All graphs g in $\mathcal{D}$ s.t. $\lambda(g,q) \leq l$

```
1. for each graph g ∈ D do
2.    if L_m(g, q) > l then
3.       continue;
4.    end if
5.    if τ(g, q) ≤ l then
6.       report g as a result;
7.       continue;
8.    end if
9.    if ρ(g, q) ≤ l then
10.      report g as a result;
11.      continue;
12.   end if
13.   if λ(g, q) ≤ l then
14.      report g as a result;
15.   end if
16. end for
```

5.2.1 Exact Subgraph Search

According to Lemma 2.2, if g_1 is subgraph isomorphic to g_2, no vertex relabelling will exists in the optimal alignment that make g_2 reach g_1. When vertex relabelling is not allowed, the edit distance between two star structures is therefore redefined (recall from Lemma 4.1) as follows:

$$\lambda'(s_1, s_2) = T'(s_1, s_2) + d(L_1, L_2)$$

where

$$T'(s_1, s_2) = \begin{cases} 2 + |L_1| + |L_2| & \text{if } l(r_1) \neq l(r_2), \\ 0 & \text{otherwise.} \end{cases}$$

Accordingly, without vertex relabelling in the edit operations, we can induce the following lemma from the analysis illustrated in Lemma 4.2:

Lemma 5.1. *Let g_1 and g_2 be two graphs, if no vertex relabelling is allowed in the edit operations, $\mu'(g_1, g_2) \leq 4 \cdot \lambda'(g_1, g_2)$, where μ' and λ' are mapping and edit distances in the case of no vertex relabelling.*

Proof. This proof can be easily obtain by considering only the first two cases shown in the proof of Lemma 4.2. □

According to Lemma 5.1, a lower bound L'_m of λ' can therefore be introduced, $\lambda' \geq L'_m = \frac{\mu'}{4}$. Note that, if g_1 is a subgraph of g_2, $\lambda(g_1, g_2) = \lambda'(g_1, g_2)$ and $\lambda(g_1, g_2) = (|E_2| - |E_1|) + (|V_2| - |V_1|)$ must hold. Therefore, L'_m can be easily applied in a filtering algorithm ExactSub for performing exact subgraph search i.e. for a query g_1 and a data graph g_2, if $L'_m > (|E_2| - |E_1|) + (|V_2| - |V_1|)$, g_2 can be safely filtered.

5.2.2 Approximate Subgraph Search

We next look at approximate subgraph search.

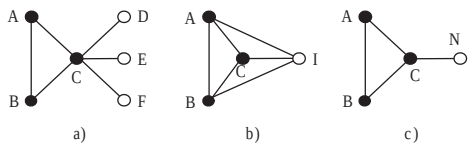


Figure 4: Example Graphs

In [35], Yan et al. introduced Grafil for performing approximate subgraph search by allowing edge relaxations. Assuming that g_3 is the maximal common subgraph between a query graph g_1 and a data graph g_2 , the number of edge relaxations in Grafil is defined as $|E_1| - |E_3|$. Note that, the definition of edge relaxation implicitly implies that no vertex relabelling is allowed. This similarity measure, however, does not take the vertex mismatches into account. For example in Figure 4, according to this measure, the number of edge relaxations between a) and c) is 3, which is the same as that of b) and c). However, a) and b) are quite different. We therefore introduce the following similarity measure based on the graph edit distance to overcome this weakness.

Definition 5.1. *A graph g_1 is said to be **θ-subgraph isomorphic** to g_2 if there exists a graph g_3 s.t. $g_3 \sqsubseteq g_2$ and $\lambda'(g_1, g_3) \leq \theta$.*

Thus, in Figure 4, a) is a 6-subgraph of c), while b) is a 4-subgraph of c). Given a query g_1 and a graph database D, the problem of ***θ-subgraph search*** is to find out all graphs g_2 in D of which g_1 is a θ-subgraph.

Furthermore, assuming $l = |E_2| - |E_1| + |V_2| - |V_1|$, then

$$\begin{aligned} \lambda'(g_1, g_2) &\leq \lambda'(g_1, g_3) + \lambda'(g_3, g_2) \\ &= |E_2| - |E_3| + |V_2| - |V_3| + \lambda'(g_1, g_3) \\ &= |E_2| - |E_1| + |V_2| - |V_1| + 2\lambda'(g_1, g_3) \\ &= l + 2\lambda'(g_1, g_3) \end{aligned}$$

Thus, if g_1 is a θ-subgraph of g_2, $\lambda'(g_1, g_2) \leq l + 2\theta$ must hold. Accordingly, we devise a filtering algorithm AppSub to perform θ-subgraph search, in which L'_m is used as a filter: if $L'_m(g_1, g_2) > l + 2\theta$, g_2 can be safely filtered.

Lemma 5.2. *If g_1 is subgraph isomorphic to g_2 within n edge relaxations, g_1 must be a $2n$-subgraph of g_2.*

Proof. We will prove by induction.

Let g_3 be the maximal common subgraph of g_1 and g_2, we will show that if $|V_1| - |V_3| = k$, at least k edges in $E_1 - E_3$ are needed to ensure that g_1 connected. If $|V_1| - |V_3| = 1$, apparently one such edge is needed to do so. Assume the above statement is true when $k = i - 1$. In the case of $k = i$, at least one edge is needed to make the newly introduced vertex connects to the other vertices. As such, k edges are therefore needed in the case of $k = i$. Assume $|V_1| - |V_3| = n + 1$, then at least $n + 1$ edges exist in $E_1 - E_3$. However, g_1 is a subgraph of g_2 within n edge relaxations, $|E_1| - |E_3| \leq n$ holds. Thus, $|V_1| - |V_3| \leq n$ and $\lambda'(g_1, g_3) \leq 2n$. g_1 is therefore $2n$-subgraph isomorphic to g_2. □

According to the above lemma, for the same query q and graph database D, the result set returned by performing $2n$-subgraph search is a superset of the result set returned by performing Grafil within n edge relaxations. Later on in the experiment, we will show that using $2n$-subgraph search will in fact return much less candidates for exact edge relaxation computation compare to the greedy filter approach that is adopted by Grafil.

To our best knowledge, almost all existing algorithms of subgraph search adopted the feature-based indexing framework which require the search for features which can involve expensive data mining processes. ExactSub and AppSub that are introduced here do not need index and can filter graphs without performing pairwise subgraph isomorphism determination. In addition, ExactSub and AppSub inherently support both two kinds of subgraph search, i.e.,

traditional subgraph search[37] and containment search[8]. In comparison to traditional subgraph search that retrieves all the graphs containing the query q, containment search returns all the graphs contained by q. However, for existing subgraph search systems, two distinct index structures must be maintained to support these two kinds of subgraph search[8]. The performance of ExactSub and AppSub are investigated in Section 6.

6. EXPERIMENTAL STUDY

In this section, we present our experimental study on both real and synthetic datasets. We first compared three methods for obtaining lower and upper bounds of λ with the exact graph edit distance computation algorithm. After that, experiments were conducted to evaluate the scalability of these methods in terms of the number of graphs and the size of graphs. Finally, a variety of experiments were also conducted to examine the performance of the three graph search algorithms which apply these bounds, i.e., AppFull, ExactSub and AppSub.

Parameter	Description
D	the number of graphs produced
T	average graph size
V	the number of vertex labels used

Table 2: Parameters of Synthetic Data Generator

In our study, all experiments were performed on a 2.40GHz Inter(R) Pentinum(R) PC with 512MB of main memory, running Debian Linux. All programs were implemented in C++ using the GNU g++ compiler with -O2 optimization. Two kinds of datasets were used through our experimental study: one real dataset and a series of synthetic datasets.

Real dataset. The real dataset used here is the AIDS antivirus screen compound dataset from the Developmental Theroapeutics Program in NCI/NIH which is available publicly[5]. It has been widely used in a large number of existing studies[35, 8, 37], and contains 42,687 chemical compounds, among which 422 of them belong to CA, 1081 are of CM and the remaining are in class CI.

Synthetic datasets. Synthetic datasets were produced by a graph data generator kindly provided by Kuramochi and Karypis. This generator allows the user to specify various parameters, and only three of them related to our experiments are shown in Table 2. For other parameters, we used the default values provided by this generator. For more details about this generator please refer to [19].

6.1 Comparison with the Exact Algorithm

We first conducted experiments to compare the runtime of four algorithms for computing L_m, τ, ρ and λ, where λ is provided by an exact graph edit distance computation algorithm Exact based on the well-known A* algorithm. As stated in [22], Exact is only able to compute the edit distances of graphs typically containing 12 vertices at most in practice. Accordingly, 1000 graphs were produced by the synthetic generator by setting D=1k, T=10 and V=4. From these 1000 graphs, ten graphs each of which contains 10 vertices were randomly selected to form the graph database $\mathcal{D}$. Meanwhile, six query groups were constructed each of which contains 10 graphs. All graphs in the same query group have the same number of vertices, and the number of vertices residing in each graph among different groups varies from 5 to 10.

[5]http://dtp.nci.nih.gov/docs/aids/aids_data.html

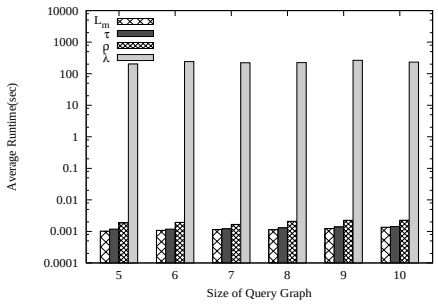
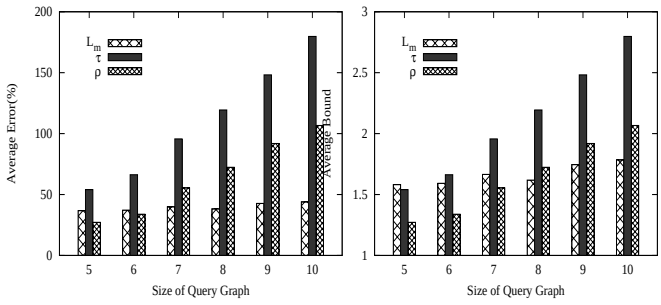


Figure 5: Runtime Comparison with Exact

Figure 5 depicts the average runtime for calculating four different distances between a query graph and the graph database $\mathcal{D}$. The X-axis shows the number of vertices contained in the query graph, and the Y-axis shows the corresponding average runtime in log scale. From this figure we can observe that the computation of λ is much more expensive than that of the other three distances, and the other three algorithms are more than 10,000 times faster than Exact.



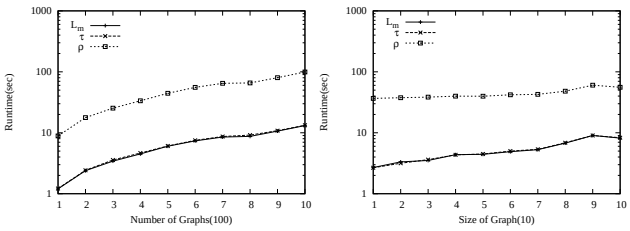
a) e_1 b) e_2

Figure 6: Comparison with Exact

Besides the comparison in terms of the runtime, we also conducted experiments to show which one of L_m, τ and ρ is tighter to λ. In order to measure the tightness, two measures were introduced. The first one is e_1 defined as $\frac{|d-\lambda|}{\lambda} \times 100\%$, where d is one of L_m, τ and ρ. However, this measure has a problem that the field of e_1 over L_m is [0,1] while the field of e_1 over τ and ρ is [0,+∞]. Thus, we introduced the second measure e_2 which is defined as $max\{\frac{d}{\lambda}, \frac{\lambda}{d}\}$. Actually, e_2 is popularly adopted to measure the approximation ratios for approximation algorithms. Figure 6 depicts the data of e_1 and e_2 respectively. From this figure we can see that ρ is always tighter to λ than τ which is consistent with the theoretical analysis.

6.2 Scalability Study

We then conducted experiments to evaluate the scalability of three bounding algorithms in terms of the number of graphs and the size of graphs.



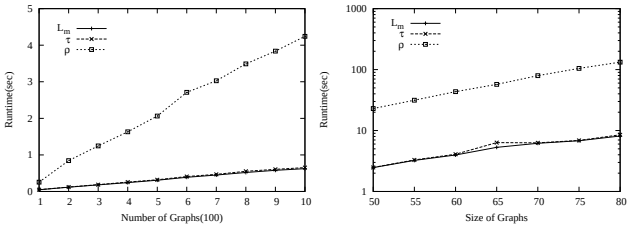
a) number of graphs b) size of graphs

Figure 7: Scalability over Synthetic Datasets

6.2.1 Scalability over Synthetic Dataset

On the one hand, we conducted experiments to evaluate their scalability over synthetic datasets. First, the scalability in terms of the number of graphs were examined. By setting T=80,V=50 and varying D from 100 to 1000, a series of synthetic datasets were produced as graph databases. And then, 10 query graphs were generated by setting D=10 to compose a query group. Figure 7 a) shows the average runtime for computing L_m, τ and ρ over graph databases with different cardinalities. Based on this figure, we can observe that all three bounding algorithms show good scalability in terms of the number of graphs. Second, experiments were conducted to evaluate their scalability in terms of the size of graphs. By setting D=10, V=50 and varying T from 50 to 100 in steps of 10, six query groups were produced each of which contains 10 graphs. At the same time, a synthetic dataset containing 1000 graphs were generated as the graph database. Figure 7 b) illustrates the total runtime for calculating different distances between the query group and the graph database. From Figure 7 b) we can see that these three algorithms also have good scalability in terms of the size of graphs.

6.2.2 Scalability over AIDS Dataset



a) number of graphs b) size of graphs

Figure 8: Scalability over AIDS Datasets

On the other hand, we also examined their scalability over the real dataset. First, 10 graphs were randomly selected from the AIDS dataset as query graphs, and a series of graph databases were generated by randomly choosing specific number of graphs from the AIDS dataset. Figure 8 a) illustrates the average runtime for computing different distances for a query graph in different graph databases. As shown in Figure 8, all these three algorithms scale linearly in terms of the number of graphs over AIDS dataset. After that, by varying the size of query graphs, experiments were conducted to evaluate the scalability in terms of the size of graphs. In this case, 1000 graphs were randomly selected from AIDS dataset as the graph database. For each query group, it contains 10 graphs containing the same number of vertices, which were also randomly selected from the AIDS dataset. Figure 8 b) depicts the average runtime from which we can observe that these bounding algorithms also scale linearly in terms of the size of graphs over real dataset.

Consequently, all three bounding algorithms have good scalability in terms of the number of graphs and the size of graphs over both synthetic and real datasets.

6.3 Graph Search Performance

Having examined the scalability of bounding algorithm, we then investigate the performance of three graph search algorithms applying these bounds.

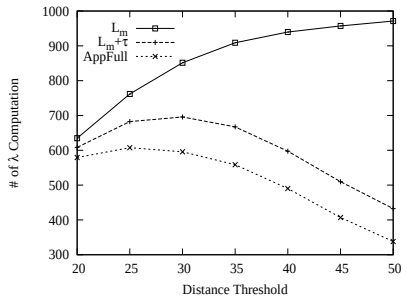
6.3.1 Approximate Full Graph Search Performance

Applying L_m, τ and ρ, AppFull shown in Algorithm 3 is introduced to perform approximate full graph search over graph databases. In this subsection, we investigate its performance in terms of the number of expensive λ computations. In order to compare the effectiveness of different bounds, we implemented two variants of AppFull by exploiting different bounds. For instance, the legend "L_m" denotes a variant of AppFull using only L_m without τ and ρ, the legend "L_m+τ" denotes a variant of AppFull using L_m and τ but without ρ. The results are depicted in Figure 9, the X-axis shows the distance thresholds used in the search, and the Y-axis shows the average number of expensive λ computations incurred in different algorithms. Apparently, it is always preferable to filter as many graphs as possible before performing the expensive λ computation.

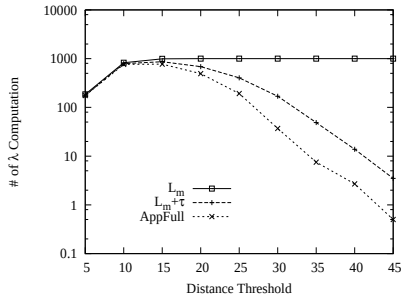
For the experiments conducted on the real dataset, 10 and 1000 graphs were randomly selected from AIDS dataset to form query graphs and the graph database respectively. While for synthetic datasets, the query graphs and the graph database were produced by the synthetic generator using parameters D=10 and D=1000 respectively. From Figure 9 we can see that AppFull outperformed the other two variants, i.e., the application of upper bounds make AppFull effective. In addition, AppFull also outperformed "L_m+τ", i.e., the introduction of ρ makes AppFull more effective. For example, running on the AIDS dataset with distance threshold 50, about 970 and 432 λ computations were needed in "L_m" and "L_m+τ" respectively, while in AppFull only 330 λ computations were needed. In addition, the runtime of AppFull is negligible in comparison with the expensive accurate edit distance computation. Without computing λ, AppFull takes less than 1 second per query on both real and synthetic graph databases.

6.3.2 Exact Subgraph Search Performance

One of important applications of the lower bound of λ is to serve as a filter for performing subgraph search over graph databases. We first evaluated the performance of ExactSub in terms of the number of candidate graphs returned. Actually, we should compare the performance of ExactSub with other exact subgraph search systems, such as *cIndex*[8] and *Tree+*Δ[37]. However, because these approaches involve complicated modules, such as subgraph pattern mining, feature selection and index construction, all of them are still in process for release. Meanwhile, with these complicated modules, our own implementations also cannot guarantee to

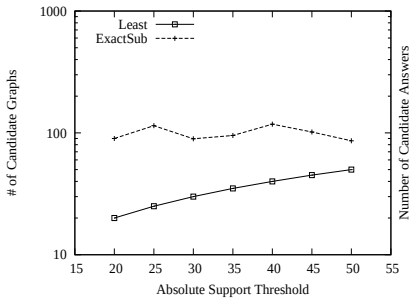


a) AIDS dataset

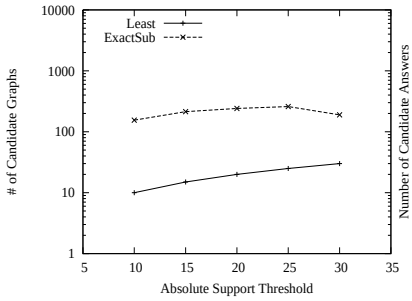


b) Synthetic dataset

Figure 9: AppFull

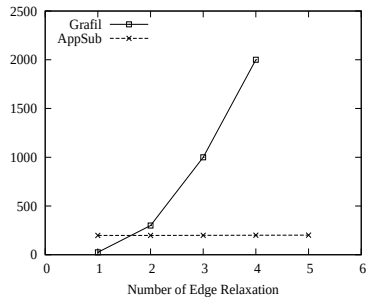


a) AIDS dataset

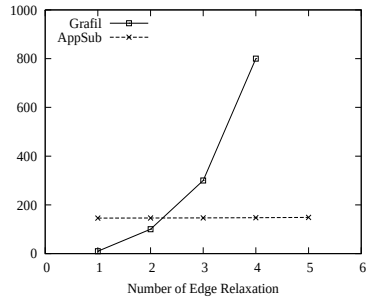


b) Synthetic dataset

Figure 10: ExactSub



a) Q_{16}



b) Q_{20}

Figure 11: AppSub

achieve the good performance shown in their papers. Therefore, we did not compare with them. But we believe that our experimental results here are adequate enough to show the performance of ExactSub.

In the experiments, 1000 graphs were randomly selected from AIDS dataset as the real graph database D_r, and 1000 graphs were produced by the synthetic generator as the synthetic graph database D_s. In order to get appropriate subgraphs as query graphs, we used gSpan[33] to mine frequent subgraphs from D_r and D_s. By setting min_sup=20, 9,263 frequent subgraphs were discovered from D_r, while 2,252 frequent subgraphs were discovered from D_s with min_sup=10. From these frequent subgraphs, we randomly chose 10 graphs with the same absolute support to form a query group. For example, for the AIDS dataset, by ranging sup from 20 to 50, 7 query groups were produced. While for the synthetic dataset, 5 query groups were constructed by ranging sup from 10 to 30. The reason for choosing such lower supports is that if the support is higher, the number of frequent subgraphs with this support might be fewer, e.g., only 1 or 2 such subgraphs. Figure 10 depicts the performance of ExactSub running on both synthetic and real datasets. The Y-axis shows the average number of candidate graphs returned. For a query graph with sup=n, there are n super-graphs of this query graph in the graph database. In addition, because we do not take the edge labels into account in our study , the real number of such super-graphs should be greater than n. The legend "Least" in Figure 10 indicates the least number of super-graphs in the graph databases which can never be saved by any filtering efforts. From Figure 10 we can observe that ExactSub is effective in filtering irrelevant graphs over both synthetic and real datasets. For example, using the query graphs with sup=30 over the AIDS dataset, ExactSub can filter about 91% of the graphs in the graph databases. The effectiveness of ExactSub also confirms the effectiveness of L'_m in filtering irrelevant graphs.

6.3.3 Approximate Subgraph Search Performance

At last, the performance of AppSub was compared with Grafil[35] in terms of the number of candidate graphs. Because the software package of Grafil involves many pieces of codes from different sources, it is still in process for release. We therefore cannot conduct a direct comparison. Instead, we exploited the same experimental settings used in [35]. 10,000 graphs were randomly selected from the AIDS dataset to form the graph database, and the query graphs were directly sampled from the database and were grouped together according to their edge numbers. The query group is denoted by Q_m where m is the edge number of the graphs in Q_m. Figure 11 depicts the performance comparison between AppSub and Grafil w.r.t. two query sets Q_{16} and Q_{20} (the data of Grafil were directly copied from figure 13 and figure 14 in [35]). The Y-axis shows the average number of candidate graphs returned by each algorithm, and the X-axis shows the edge relaxation ratio used in Grafil. To compare the performance of Grafil and AppSub clearly, we doubled the approximation ratio used in AppSub. For example, if 2 edges is used in Grafil as the edge relaxation ratio, AppSub performed the 4-subgraph search correspondingly.

Based on Figure 11, even the result set returned by performing $2n$-subgraph search is a superset of the result set returned by performing Grafil within n edge relaxations, AppSub outperformed Grafil significantly when $n > 2$ by a factor of 5-10 times.

7. CONCLUSION AND DISCUSSION

In this paper, three novel distances, L_m, τ and ρ, are introduced to lower and upper bound the graph edit distance λ in polynomial time. The relationships of them can be represented by the inequality $L_m \leq \lambda \leq \rho \leq \tau$, and their com-

putation complexities are shown in Table 3. In comparison to the exact computation of graph edit distance, these three bounding algorithms are more efficient and have good scalability in terms of the number of graphs and the size of graphs. Applying these efficiently computable bounds, three effective algorithms AppFull, ExactSub and AppSub are proposed for to perform various kinds of graph searching. The good performance of these graph search algorithms demonstrated by the experimental results also confirms the effectiveness of these three bounds.

Distance	Complexity	Distance	Complexity
L_m	$\Theta(n^3)$	τ	$\Theta(n^3)$
ρ	$O(n^6)$	λ	NP-Hard

Table 3: Complexity of Different Distances

So far, our discussion focuses on undirected graphs. Fortunately, our proposed algorithms can be easily extended to directed graphs. For directed graphs, a directed star structure $\vec{s}$ can be represented by a 4-tuple $\vec{s} = (r, I, O, l)$, where I is the set of vertices from which there are edges to r and O is the set of vertices to which there are edges from r. Accordingly, star edit distance can be newly defined as

$$\lambda(\vec{s}_1, \vec{s}_2) = T(r_1, r_2) + d(I_1, I_2) + d(O_1, O_2)$$

Based on this new definition, L_m, τ and ρ can be easily applied to directed graphs.

8. REFERENCES

[1] H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Trans. PAMI*, 15(5):522–525, 1993.

[2] V. Arvind and P. P. Kurur. Graph isomorphism is in spp. *Information and Computation*, 204(5):835–852, 2006.

[3] S. Berretti, A. D. Bimbo, and E. Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Trans. PAMI*, 23(10):1089–1105, 2001.

[4] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM '02*.

[5] J. M. Bower and H. Bolouri. *Computational Modeling of Genetic and Biochemical Networks (Computational Molecular Biology)*. 2004.

[6] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web:experiments and models. In *WWW '00*.

[7] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[8] C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang, and X. Gu. Towards graph containment search and indexing. In *VLDB '07*.

[9] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-definable counting classes. *J. Comput. Syst. Sci.*, 48(1):116–148, 1994.

[10] M.-L. Fernández and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753–758, 2001.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. 1990.

[12] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. SSC*, 4(2):100–107, 1968.

[13] H. He and A. Singh. Closure-tree: An index structure for graph queries. *ICDE'06*.

[14] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD '07*.

[15] H. Hu, X.Y., Y. Hang, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological network for functional discovery. *Bioinformatics*, 1(1):1–9, 2005.

[16] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. PAMI*, 28(8):1200–1214, 2006.

[17] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB '05*.

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.

[19] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM '01*.

[20] B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Trans. PAMI*, 20(5):493–504, 1998.

[21] R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian Graph Edit Distance. *IEEE Trans. PAMI*, 22(6):628–635, 2000.

[22] M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *SSSPR '06*.

[23] E. G. M. Petrakis and C. Faloutsos. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):435–447, 1997.

[24] J. Raymond, E. Gardiner, and P. Willett. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *The Computer Journal*, 45(6):631–644, 2002.

[25] K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *MLG '07*.

[26] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS '02*.

[27] S. Srinivasa and S. Kumar. A platform based on the multi-dimensional data modal for analysis of bio-molecular structures. In *VLDB '03*.

[28] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. Saga: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2007.

[29] N. Trinajstic, J. V. Knop, W. R. Muller, K. Syzmanski, and S. Nikolic. *Computational Chemical Graph Theory: Characterization, Enumeration and Generation of Chemical Structures by Computer Methods*. 1991.

[30] S. Trissl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD '07*.

[31] P. Willett, J. Barnard, and G. Downs. Chemical similarity searching. *J. Chem. Inf. Comput. Sci*, 38(6):983–996, 1998.

[32] R. C. Wilson and E. R. Hancock. Structural matching by discrete relaxation. *IEEE Trans. PAMI*, 19(6):634–648, 1997.

[33] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM'02*.

[34] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD '04*.

[35] X. Yan, F. Zhu, P. S. Yu, and J. Han. Feature-based similarity search in graph structures. *ACM TODS*, 31(4), 2006.

[36] R. Yang, P. Kalnis, and A. K. H. Tung. Similarity evaluation on tree-structured data. In *SIGMOD '05*.

[37] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB '07*.