Efficient Graph Similarity Search Over Large Graph Databases

Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao

Abstract—Since many graph data are often noisy and incomplete in real applications, it has become increasingly important to retrieve graphs g in the graph database D that approximately match the query graph q, rather than exact graph matching. In this paper, we study the problem of graph similarity search, which retrieves graphs that are similar to a given query graph under the constraint of graph edit distance. We propose a systematic method for edit-distance based similarity search problem. Specifically, we derive two lower bounds, i.e., partition-based and branch-based bounds, from different perspectives. More importantly, a hybrid lower bound incorporating both ideas of the two lower bounds is proposed, which is theoretically proved to have higher (at least not lower) pruning power than using the two lower bounds together. We also present a uniform index structure, namely u-tree, to facilitate effective pruning and efficient query processing. Extensive experiments confirm that our proposed approach outperforms the existing approaches significantly, in terms of both the pruning power and query response time.

Index Terms—Graph edit distance, lower bound, graph database, graph similarity search

1 Introduction

Recently, graph data models have attracted increasing research interest, because many data in various applications can be represented by graphs, such as chemical compounds [1], social networks [2], road networks [3], and Semantic Web [4]. The growing popularity of graph data requires efficient graph data management techniques. Thus, many queries have been investigated, such as shortest path query [5], [6], reachability query [7], [8], [9], and (sub)graph query [10], [11], [12]. Among these, (sub)graph query (i.e., given a query graph q, finding all graphs g in a graph database D, such that q is (sub)graph isomorphic to g) has been well studied.

However, some real-life graphs, such as protein-protein-interaction networks [13], often contain noises. It is desirable to find a robust solution to retrieve graphs that are of interest to users even in the presence of noises and errors. An interesting topic is to study graph similarity search, which retrieves all graphs g from a database D that approximately match with q under some similarity measure. A number of graph similarity measures have been proposed [14], [15], [16], [17], [18], [19], among which, two classical graph similarity functions are *maximum common subgraph* (MCS) [14] and *minimum graph edit distance* (MGED) [17]. Note that the two measures are inter-related [20] and we focus on the MGED in this paper. As a widely used structural similarity measure, MGED is defined as the minimum operation cost (addition, deletion, and substitution) of transforming from one graph q to another graph g (Definition 2.2). MGED is a flexible graph similarity measure and it has been used in many applications, such as graph classification [21], graph clustering [22], object recognition in computer vision [23], and molecule comparison in chemistry [24].

In this paper, based on MGED, we study the problem of *graph similarity search*: Given a graph database D, a query graph q, and a threshold τ, the goal is to find some graphs g in D, such that $MGED(q, g) \leq \tau$. Before presenting our method, we first demonstrate the usefulness of graph similarity search by the following motivation example.

Motivation example. Fig. 1 shows a chemical compound database D containing compounds in the graph representation whose properties have been well studied. When we study the properties of a new compound q, we can issue a *graph similarity search query* to retrieve the compounds (in D) that have similar structures to q. We often call this step "compound screening [25]" in the drug development. According to "structure-activity relationship (SAR)",[1] a molecule's biological activity is often determined by its chemical structure [26]. Therefore, it is reasonable to assume that q may have similar biological activities to a graph g, if g has similar structure to q. Although we still need to verify these candidates by subsequent chemical and biological experiments, the compound screening in the first step can help save a lot of costs. In other words, the graph similarity search can provide the starting point for understanding the new compound, which plays a very important role in the drug discovery.

Furthermore, graph similarity search can also find applications on structural pattern recognition, such as logo image

- *W. Zheng, L. Zou, D. Wang, and D. Zhao are with Peking University, Beijing 100080, China. E-mail: {zhengweiguo, zoulei, wangd, zhaody}@pku.edu.cn.*
- *X. Lian is with the University of Texas-Pan American, Edinburg, TX 78539. E-mail: lianx@utpa.edu.*

Manuscript received 25 Oct. 2013; revised 25 July 2014; accepted 28 July 2014. Date of publication 19 Aug. 2014; date of current version 3 Mar. 2015. Recommended for acceptance by R. Jin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2349924

1. The structure-activity relationship (SAR) is the relationship between the chemical structure of a molecule and its biological activity. The analysis of SAR enables the determination of the chemical groups responsible for evoking a target biological effect in the organism [26].

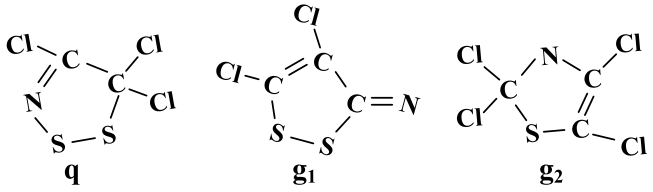


Fig. 1. A query graph and two sample data graphs.

search [22]. Due to the invariability to rotation and translation of images, graphs are widely applied to represent objects. Thus, pattern recognition becomes a problem of graph matching. In order to be error-tolerant to noises, MGED-based graph matching is often used in pattern recognition [23].

The motivation example above illustrates the importance of the graph similarity search problem. However, it is not a trivial task. Since computing MGED is an NP-hard problem [17], most existing solutions adopt the filter-and-refine framework to speed up the query processing, that is, first using an effective and efficient pruning strategy to filter out as many false positives (graphs that are impossible in the results) as possible, and then validating the remaining candidates by computing graph edit distance. In general, the existing pruning rules can be divided into two categories: the *global filter* and the *n-gram* based filter. With these filters, the data graphs whose lower bounds are larger than a user-specified threshold τ can be filtered out safely.

1) Global filter. There are two existing global filters. The first one is to utilize the difference of the vertex/edge number as the lower bound [17]. The second one considers the difference of vertex/edge labels to further improve the pruning power [27]. Since these methods do not employ the graph structure, the lower bounds are not tight enough.

2) n-gram based filter. The other category of filters adopt the *n-gram* method, which is used in string similarity search problem [28], [29], [30]. The basic idea of these methods is to select some subgraph structures as the *n-grams*, and then derive the lower bound based on the common *n*-grams of the two graphs. Wang et al. [31] propose *k-Adjacent Tree* (*k-AT*) algorithm, which defines an n-gram as a tree consisting of a vertex v and paths starting from v with lengths no larger than k. Apparently, a single edit operation may affect many *k-AT* trees, especially when k is larger than 2. The c-star structure used in [17] is exactly the same as *k-AT* when $k = 1$. Specifically, the c-star based lower bound in [17] is $L_m(g_1, g_2) = \frac{\mu(g_1,g_2)}{max\{4,\lceil max\{\delta(g_1),\delta(g_2)\}\rceil+1\}}$, where $\mu(g_1, g_2)$ is the mapping distance between g_1 and g_2 according to the bipartite graph, and $\delta(g_1)$ and $\delta(g_2)$ are the maximum degrees in g_1 and g_2, respectively. According to the equation of the lower bound, if g_1 or g_2 has a high-degree vertex, the lower bound might be very small. Similar to *k-AT*, Zhao et al. [27] compute the lower bound by employing the *path-based* n-grams. However, these *path-based* n-grams still share many overlapping structures, if there are some high-degree vertices. Therefore, in such a case, the lower bound computed by the path-based *n*-grams can be rather loose.

Generally speaking, there are two problems with the existing n-gram based pruning methods: 1) the lower bound is not tight, since the existing n-grams have many overlaps and a single edit operation may affect many n-grams; and 2) it is very costly to compute the lower bound. For example, the time complexity of computing the star-based lower bound is cubic time [17].

Considering the limitations above, we study MGED-based graph similarity search problem systematically in this paper. Given a query graph q and a data graph g, we first derive two novel lower bounds—disjoint-partition lower bound $dist_P(q, g)$ and branch-based lower bound $\lambda(q, g)$—that are independent of each other. In order to further improve the pruning power, based on the two lower bounds, we propose a new hybrid lower bound $dist_H(q, g)$. More importantly, we theoretically prove that $dist_H(q, g) \geq MAX(dist_P(q, g), \lambda(q, g))$.

The basic idea of the disjoint-partition lower bound $dist_P(q, g)$ is: given a query graph q, a data graph g, and a threshold τ, if we can find $(\tau + 1)$ *mismatching substructures*[2] in q and any two of them have no overlaps, the lower bound is $dist_P(q, g) = \tau + 1 > \tau$, meaning that g can be filtered out safely. The basic intuition is that one edit operation can only affect one mismatching substructure, since any two of mismatching substructures have no overlaps. Therefore, we can conclude that it must require at least $(\tau + 1)$ edit operations to transform from graph q to graph g. It also implies that g is impossible to be a similarity match of q (as the threshold is τ), and g can be pruned safely. The key challenge of this method is how to determine whether there are $(\tau + 1)$ disjoint mismatching substructures, once q and g are given. Unfortunately, it is at least as hard as an NP-hard problem. A heuristic on-demand solution is developed in this paper.

The branch-based lower bound $\lambda(q, g)$ follows the n-gram approach. However, we propose a different n-gram, namely *branch*, which is defined as a structure consisting of one vertex and its adjacent edges without including the other endpoints (the formal definition of *branch* will be presented in Section 4). The superiority of *branch* lies in that*a single edit operation (addition, deletion, or substitution) can affect only two branches at most*. Although a branch is structural similar to c-star [17] except for excluding the leaf nodes of a c-star, one edit operation can affect $MAX(\delta(q), \delta(g))$ c-stars. If a query graph or a data graph has some high-degree vertices, the lower bound in c-star is very loose due to the large penalty ratio in c-star lower bound equation. Furthermore, unlike the cubic time complexity in the c-star method, we design *an efficient algorithm with the time complexity* $O(|V| \cdot \log|V|)$ to compute the branch-based lower bound.

To further improve the pruning power, we propose a hybrid lower bound. Given a query graph q and a data graph g, we first find k mismatching substructures in q. Obviously, if $k \geq (\tau + 1)$, data graph g can be filtered out safely. We only consider the case that $k \leq \tau$. As we know, each mismatching substructure requires at least one edit operation. We enumerate all possible one-step edit operations over the mismatching substructures by introducing a "wildcard" label to transform q to q^*. We can prove that $dist_H(q, g) = k + \lambda(q^*, g)$ is a lower bound for $MGED(q, g)$ and $dist_H(q, g) \geq MAX(dist_P(q, g), \lambda(q, g))$. In other words, $dist_H(q, g)$ provides stronger or at least not lower pruning power than using $dist_P(q, g)$ and $\lambda(q, g)$ together.

2. If a substructure of q does not occur in graph g, the substructure is called a mismatching substructure.

TABLE 1
Frequently-Used Notations

Notation	Definition and Description
q (or g)	the query graph (or data graph)
$V(g)$	the vertices of graph g
$E(g)$	the edges of graph g
D	the graph database consisting of g
b	a branch structure of g
τ	the threshold of graph edit distance
$mged(q, g)$	the graph edit distance between q and g
$bed(b_1, b_2)$	the branch distance between b_1 and g_2
$B(g)$	the multiset of branches in graph g
$\lambda(q, g)$	the mapping distance of $B(q)$ and $B(g)$
$dist_P(q, g)$	the partition-based lower bound
$MS(q, g)$	the mismatching structure multiset of q over g
$dist_B(q, g)$	the branch-based lower bound
$dist_{CB}(q, g)$	the compact branch lower bound
$dist_H(q, g)$	the hybrid lower bound

In summary, we make the following contributions.

- We propose two lower bounds from different perspectives: one is based on the mismatching structures and the other one is based on the new n-grams. The two lower bounds are independent of each other.
- Based on the ideas of the two lower bounds, we design a new hybrid lower bound rather than simply checking the two lower bounds one by one. More importantly, we theoretically prove that the hybrid lower bound is no smaller than the two lower bounds.
- In order to reduce the search space, we carefully devise a uniform index structure, namely *u-tree*, to speed up filtering process.
- Extensive experiments over both real and synthetic graphs confirm the effectiveness and efficiency of our proposed approaches.

Organization. The problem definition is introduced in Section 2. Section 3 presents techniques of the partition-based lower bound, followed by the strategies of branch-based lower bound in Section 4. A tighter hybrid filter is proposed in Section 5. Section 6 describes the *u-tree* index and query processing. Experimental results are reported in Section 7. Section 8 investigates the research work related to this paper. Finally, Section 9 concludes this paper.

2 Background and Problem Definition

In this section, we first formally define our problem. Table 1 lists the frequently-used notations in this paper.

For the ease of presentation, we consider a simple undirected attributed graph in this paper. A simple graph means that it does not contain self-loops or multi-edges. Formally, a simple undirected attributed graph is defined as a six-tuple $g = (V, E, L_V, L_E, \Sigma_V, \Sigma_E)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, Σ_V and Σ_E are the label sets of V and E, respectively, and L_V and L_E are label functions that assign labels to vertices and edges, respectively. Note that our solution can be easily extended to directed graphs without loss of generality.

Definition 2.1 (Graph Isomorphism). *Graph g_1 is graph isomorphic to graph g_2 if there exists a bijective function f: $V(g_1) \rightarrow V(g_2)$ s.t. 1) $\forall v \in V(g_1)$, $f(v) \in V(g_2) \wedge L_V(v) = L_V(f(v))$, and 2) $\forall e(v_1, v_2) \in E(g_1)$, $e(f(v_1), f(v_2)) \in E(g_2) \wedge L_E(e(v_1, v_2)) = L_E(e(f(v_1), f(v_2)))$.*

Let $g_1 = g_2$ denote that two graphs g_1 and g_2 are graph isomorphic to each other.

There are six primitive edit operations on graphs [17]: insert/delete an isolated vertex with label, substitute a/an vertex/edge label, and insert/delete an edge between two vertices. Given two graphs g_1 and g_2, there exists a sequence of primitive edit operations to transform g_1 to g_2, such as, $g_1 = g_1^0 \rightarrow g_1^1 \rightarrow \cdots \rightarrow g_1^k = g_2$. We may have different operation sequences to transform g_1 to g_2. The minimum graph edit distance (dissimilarity) between two graphs is measured by the shortest operation sequence length, as defined as follows.

Definition 2.2 (Minimum Graph Edit Distance). *Given two graphs g_1 and g_2, their minimum graph edit distance is defined as the minimum number of primitive operations needed to transform g_1 to g_1', s.t., $g_1' = g_2$, denoted by $mged(g_1, g_2)$.*

Given the definition of minimum graph edit distance (or called graph edit distance if there is no ambiguity in the context), we formalize the problem of this paper as follows.

Problem Statement (Graph Similarity Search). *Given a database consisting of $|D|$ graphs, $\mathrm{D} = \{g_1, g_2, \ldots, g_{|D|}\}$, a query graph q, and a distance threshold τ, find all graphs $g_i \in D$ s.t. $mged(q, g_i) \leq \tau$, where $mged(q, g_i)$ is defined in Definition 2.2.*

Example 1. Fig. 1 shows a query graph q and two data graphs g_1 and g_2, where vertices represent atom symbols and edges are chemical bonds. $mged(q, g_1) = 7$, $mged(q, g_2) = 8$. Neither g_1 nor g_2 is an answer to the graph similarity search with $\tau = 4$, since graph edit distances of both graphs to q are larger than τ.

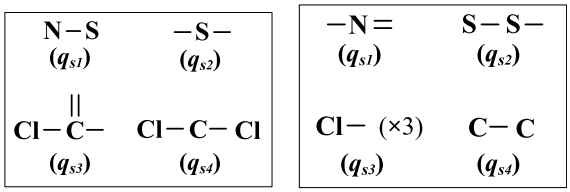
Most existing graph similarity search algorithms follow the filter-and-verification framework. In the filtering phase, we compute the lower bounds of graph edit distance between query graph q and each data graph g_i in graph database D, $i = 1, \ldots, |D|$. If the lower bound is larger than threshold τ, we can prune the data graph safely. Then, we compute the graph edit distances over candidates to find the true answers. Clearly, it is critical to efficiently estimate the lower bound as tight as possible.

We first propose two lower bounds, i.e., $dist_P(q, g)$ and $\lambda(q, g)$, from two different perspectives in Sections 3 and 4, respectively. Then, in order to further improve the pruning power, based on the two lower bounds, we propose a hybrid lower bound $dist_H(q, g)$ (in Section 5) rather than checking the two bounds one by one. We also prove that the hybrid lower bound can provide higher (at least not lower) pruning ability than using $dist_P(q, g)$ and $\lambda(q, g)$ together.

3 Disjoint-Partition Based Filter

3.1 Pruning Strategy: A Basic Idea

Definition 3.1 (Mismatching Structure). *Given a query graph q and a data graph g, a* mismatching structure *in query*



(a) partitioning results for g_1 (b) partitioning results for g_2

Fig. 2. Partitioning of the query graph for g_1 and g_2.

graph q with regard to data graph g is a subgraph (of q) that is not subgraph isomorphic to g.

For example, the subgraph "N−S" of query q in Fig. 1 does not exist in data graph g_1 in the running example. So, "N-S" is a mismatching structure with regard to g_1.

Intuitively, for any *mismatching structure* (Definition 3.1) in q, it requires at least one edit operation to transform the mismatching structure to the corresponding structure in g. Obviously, if we can find k *disjoint* (without any common vertex or edge) mismatching structures in q, k is a lower bound of the graph edit distance between q and g, as one edit operation only affects one mismatching structure. If k is larger than the threshold τ, g can be pruned safely.

Definition 3.2 (Disjoint Partition). *Given a query graph q, a disjoint partition of q is denoted as $P(q) = \{q_{s_1}, \ldots, q_{s_n}\}$, where (1) each q_{s_i} is a subgraph of q, $1 \leq i \leq n$; (2) for any two different subgraphs q_{s_i} and q_{s_j} in $P(q)$, $1 \leq i \neq j \leq n$, they do not share any vertex or edge; (3) the assembly of all subgraphs in $P(q)$ forms the query graph q.*

For example, Fig. 2a shows a disjoint-partition of q. Note that a subgraph in $P(q)$ may have some edges without endpoints or with a single endpoint. For example, "–S–" in Fig. 2a has two edges with a single endpoint S. Any two subgraphs in Fig. 2a share no common vertices or edges. Fig. 2b shows another disjoint-partition.

Definition 3.3 (Disjoint-Partition Lower Bound). *Given a disjoint-partition of query graph q (denoted as $P(q)$), the disjoint-partition based lower bound is $dist_P(q, g) = |MS(q, g)|$, where $MS(q, g)$ is a multiset of mismatching structures in $P(q)$ with regard to g.*

Theorem 3.1. $mged(q, g) \geq dist_P(q, g)$.

Proof. Considering each structure q_s in $MS(q, g)$, it requires at least one edit operation (addition, deletion, or substitution) to transform q_s to q_s' such that q_s' is subgraph isomorphic to g. Since the structures in $MS(q, g)$ are disjoint, all the operations are non-interacting. Therefore, $mged(q, g) \geq dist_P(q, g)$. □

Example 2. Consider the two graphs q and g_1 in Fig. 1 where $\tau = 2$. We partition q into four blocks as shown in Fig. 2a, among which neither q_{s_1} nor q_{s_4} can match any subgraph of g_1, so $dist_P(q, g_1) = 2$. Analogously, to obtain the lower bound of $mged(q, g_2)$, we partition q as shown in Fig. 2b, where q_{s_1}, q_{s_2}, and q_{s_4} are all mismatching structures. Hence, $dist_P(q, g_2) = 3 > \tau = 2$, and g_2 is pruned safely. Whereas, g_1 will pass this filter.

In Example 2, if we use the partitioning of q as shown in Fig. 2a to process data graph g_2, then $dist_P(q, g_2) = 1$ (mismatching structures only include q_{s_1}), which indicates that g_2 cannot be filtered out when $\tau = 2$. That is also to say different partitions lead to different pruning powers.

Motivated by Example 2, given a query graph q, we have to find different disjoint-partitions of q for different data graphs to maximize the pruning power. This raises two problems: First, give a query graph q and a data graph g, how to find an optimal disjoint-partition to maximize the disjoint-partition lower bound $dist_P(q, g)$ (Definition 3.3). Second, it is very costly to partition query graph q for each data graph in D. We need an effective strategy to avoid the sequential scan.

Unfortunately, we prove that finding the optimal disjoint-partition to maximize $dist_P(q, g)$ is at least as hard as an NP-complete problem in Section 3.2. Therefore, we design a heuristic lightweight algorithm to partition query graph q for different data graphs. In order to further reduce the filtering cost, we design a tree-style index to avoid the sequential scan over all data graphs in D in Section 6.

3.2 The Hardness of Finding the Optimal Partition

Given two graphs q and g, the goal is to find an optimal disjoint-partition $P(q)$ to maximize $dist_P(q, g)$. However, it is at least as hard as an NP-complete problem, which is the main result of this section. It is also the reason why we need a heuristic algorithm in Section 3.3 to partition q.

In order to enable the proof, we design a decision version of this optimization problem. We call it *k-Mismatching Partition Problem* as follows.

(Decision Problem 1). *Given a query graph q, a data graph g and a threshold τ, can we determine whether there exists (or does not exist) a disjoint-partition of q so that the disjoint partition-based lower bound is larger than τ, i.e., $dist_P(q, g) \geq \tau + 1$?*

Answering the Decision Problem 1 equals to answering whether we can find a $(\tau + 1)$-mismatching partition of q (i.e., $k = \tau + 1$) for g and the threshold τ. We only consider $k = \tau + 1$ because of the following reasons. First, if $k \leq \tau$, the partition-based lower bound will not be larger than τ (see Theorem 3.1). Second, if we can find a k-mismatching partition where $k > (\tau + 1)$, we can transform it into a $(\tau + 1)$-mismatching partition by merging some blocks.

Theorem 3.2. *Given a query graph q, a data graph g and a threshold τ, determining whether there exists (or does not exist) a k-mismatching partition of q is at least as hard as an NP-complete problem, where $k = (\tau + 1)$.*

Proof. Consider the case that $k = 1$, i.e., there is only one partition. The decision problem equals to determining whether q is subgraph isomorphic to g, which is a well-known NP-complete problem. Hence, determining whether there exists (or does not exist) a k-mismatching partition of q is at least as hard as an NP-complete problem. □

3.3 On-Demand Partition

Through the analysis in the previous sections, we have two observations. First, we cannot find a good partition of q to favor all data graphs, since different data graphs g require

different partitions of q to maximize the pruning power. Second, it is very difficult to find the optimal partition for each data graph. Therefore, we propose a heuristic on-demand partition of q, which is based on some statistics (of each data graph g) collected during the offline processing.

The basic idea is that we want to find as many small-size mismatching structures of q over data graph g as possible. If we find a mismatching structure, we remove it from q; and iterate the steps above until that $(\tau+1)$-mismatching structures are found or the remaining part of q is empty. Note that this is not an exact algorithm. It means that we may not find a $(\tau+1)$-mismatching partition of q over g even though there exists a $(\tau+1)$-mismatching partition. In this case, we cannot prune this data graph that is not a true answer. Regarding these candidates, we need to refine them further, so it does not lead to result dismissals. Our proposed partitioning strategy considers the following four kinds of structures.

1) *Size-1 structures*. A vertex or an edge in q.
2) *Size-2 structures*. A structure formed by a vertex and one of its adjacent edges in query q, denoted as *VE*.
3) *Size-3 structures*. A structure formed by two vertices and the edge between them in query q; or two edges and the vertex they share in q, denoted as *VEP*.
4) Consider structures whose size are larger than three.

Algorithm 1. $DyAdPartition(q, g, \tau)$

Input: A query graph q, a data graph g, and the threshold τ.
Output: $|MS(q,g)|$, the number of mismatching structures.

```
1  for each size-1 structure q_s in q do
2      if q_s has no any match over g then
3          add q_s into MS(q, g)
4          remove q_s from q
5  for each size-2 structure q_s in q do
6      if q_s has no any match over g then
7          add q_s into MS(q, g)
8          remove q_s from q
9  for each size-3 structure q_s in q do
10     if q_s has no any match over g then
11         add q_s into MS(q, g)
12         remove q_s from q
13 for each unused v_i ∈ V(q) do
14     extend v_i to obtain a structure q_s
15     if q_s has no any match over g then
16         add q_s into MS(q, g)
17         remove q_s from q
18         if |MS(q, g)| > τ then
19             return |MS(q, g)|
20 return |MS(q, g)|
```

Algorithm 1 presents the partitioning framework. Given a query graph q, finding $(\tau+1)$-mismatching partition of q follows the four-step pipeline. We remove all size-1 (i.e., vertex or edge),[3] size-2 (i.e., *VE*) and size-3 (i.e., *VEP*) mismatching structures from q in the first three steps, respectively. Finally, we try to find the mismatching structures whose size is larger than three by invoking the subgraph isomorphism verification algorithm. We adopt the state-expansion subgraph isomorphism verification algorithms (such as VF2 [11] and QuickSI [32]) to find the mismatching structures. A *state* refers to a partial match of q over g. In order to speed up the fourth step, we restrict the state size no larger than T (a turning parameter T with default value 8. It will be analyzed in Section 7.2).

The first three steps can be implemented efficiently by employing an inverted index that stores the corresponding substructures in the data graph g. Specifically, all size-1, size-2 and size-3 substructures (each substructure is denoted as s) in data graphs are keys. For each substructure s, it has a list of data graphs (in database D) that contains s.

Early stop strategy. In order to improve the partitioning efficiency, we propose two early stop rules. First, we can stop the partition process if we have found $(\tau+1)$ mismatching structures. Second, assume that we have found x mismatching structures of q over g in the first three steps. We remove these mismatching structures from q and the left part is denoted as q'. Since the size (the number of vertices and edges) of the mismatching structures (if any) obtained in the fourth phase must be larger than three. Thus, if $(|V(q')|+|E(q')|) < 3\cdot(\tau+1-x)$, it indicates that we cannot find $(\tau+1-x)$ mismatching structures in the final phase.

Complexity analysis. It is easy to know that the first three steps have the linear time complexity $O(|V(q)|+|E(q)|)$. The final step invokes a state-expansion based subgraph isomorphism verification algorithm (such as VF2 [11] and QuickSI [32]) to find mismatching structures. Although subgraph isomorphism verification is an NP-complete problem [33], we restrict the state size no larger than T (as discussed earlier). Therefore, the final step is also very fast in practice. The effect of the parameter T will be studied in Section 7.2.

4 Branch-Based Filter

In this section, we propose another lower bound that follows the n-gram filter strategy, which is different from the partition-based filter $dist_P(q,g)$. As mentioned earlier, although there are some existing different n-grams, such as c-stars and paths, the pruning abilities of existing n-gram filters are limited. The reason is that one edit operation may affect many n-grams, if there are some high-degree vertices in data graphs or query graphs. In order to address this problem, we propose to use the "branch" as n-gram. The most important benefit of "branch" n-gram is that one edit operation only affects two n-grams at most. Thus, it can provide more stable pruning power regardless of the vertex degrees. Furthermore, we design a lightweight algorithm with $O(|V|\log|V|)$ time complexity for computing the lower bound instead of the cubic-time complexity $O(|V|^3)$ in c-star method, where $|V| = MAX(|V(g)|, |V(q)|)$.

4.1 Branch Filter-A Basic Method

In this section, we present a basic branch-based method. Although it is not optimized for computing the bound, it illustrates the main idea of our branch-based filter.

Definition 4.1 (Branch Structure). *A branch structure b consists of a vertex v and a multiset of edge labels incident to v,*

3. Removing a single vertex (or an edge) will not lead to removing all the edges adjacent to this vertex (or the endpoints of the edge).

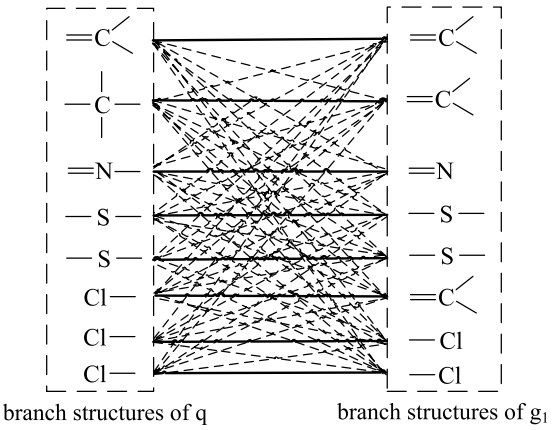


Fig. 3. Branch structures of q and g_1.

represented by $b(v) = (l_v, ES)$, where $l_v = L_V(v)$ is the label of the root vertex, and $ES = \{l_E(e) |$ edge e is adjacent to v $\}$ is the multiset of edge labels adjacent to v.

Fig. 3 shows the branch structures of graphs q and g_1 in Fig. 1. Although "branch" is similar to "c-star" [17] except for excluding the leaf nodes of a c-star, one edit operation can only affects two branches at most regardless of the degrees of the vertices. However, one edit operation can affect $MAX(\delta(q), \delta(g))$ c-stars, where $\delta(g)$ denotes the maximal vertex degree of g. Therefore, the pruning power of branch-based lower bound is much more stable than other existing n-gram bounds. According to the definition of branches, we define the distance of two branches, based on which, we derive the branch-based lower bound.

Definition 4.2 (Branch Edit Distance). *Given two vertices v_1 and v_2, their branch structures are denoted as $b_1 = (l_1, ES_1)$ and $b_2 = (l_2, ES_2)$, respectively. The branch edit distance between b_1 and b_2 is defined as follows:*

$$bed(b_1, b_2) = T(l_1, l_2) + \frac{\Gamma(ES_1, ES_2)}{2}, \tag{1}$$

where

$$T(l_1, l_2) = \begin{cases} 0, & \text{if } \ l_1 = l_2, \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

$$\Gamma(ES_1, ES_2) = max\{|ES_1|, |ES_2|\} - |ES_1 \cap ES_2|. \tag{3}$$

Given two graphs q and g, we can enumerate all branch structures of q and g to obtain two sets, $B(q)$ and $B(g)$, respectively. Thus, we can construct a bipartite graph like that in Fig. 3, where vertices represent branches and edges represent transformations between any two branches (from $B(q)$ and $B(g)$, respectively) weighted by their pairwise *branch edit distance* (defined in Definition 4.2).

Notice that if the number of branches in $B(q)$ is less than that in $B(g)$, we introduce $(|B(g)| - |B(q)|)$ blank branches in $B(q)$, and vice versa. In the following discussion, we assume that $B(q)$ and $B(g)$ have the same number of branches.

Definition 4.3. *Given two multisets of branch structures $B(q)$ and $B(g)$ with the same cardinality, the mapping distance between $B(q)$ and $B(g)$ is*

$$\lambda(q, g) = \min_{P} \sum_{b_i \in B(q)} bed(b_i, P(b_i)). \tag{4}$$

Clearly, the bijection P in Equation (4) is the minimum weighted match in the bipartite graph. Based on the distance between $B(q)$ and $B(g)$ (i.e., $\lambda(q, g)$), we can obtain a lower bound of the minimum graph edit distance between q and g (i.e., $mged(q, g)$), as shown in the following theorem.

Theorem 4.1. *Given two graphs q and g, their graph edit distance and the mapping distance between $B(q)$ and $B(g)$ satisfy the following inequality:*

$$mged(q, g) \geq dist_B(q, g) = \lambda(q, g),$$

where $B(q)$ and $B(g)$ are the branch structure multisets of q and g, respectively, and the branch structure-based lower bound is denoted as $dist_B(q, g)$.

Proof. Let $P = (p_1, p_2, \ldots, p_k)$ be an optimal alignment transforming from q to g, i.e., $k = mged(q, g)$. Accordingly, there is sequence of graph $q = q^0 \rightarrow q^1 \rightarrow \cdots \rightarrow q^k = g$, where $q^i \rightarrow q^{i+1}$ indicates transforming q^i to q^{i+1} by operation p_i. Assume that there are k_1 edge insertion/deletion/relabeling operations, k_2 vertex insertion/deletion/relabeling operations in P, then we have $k_1 + k_2 = k = mged(q, g)$.

Edge operations (insertion/deletion/relabeling). If an edge is inserted or deleted or relabeled over the graph q^i, only two branches are affected. Thus, we can know that $\lambda(q^i, q^{i+1}) \leq \frac{2}{2} = 1$ in the case of performing one edge operation (inserting/deleting/relabeling an edge) over q^i.

Vertex operations (insertion/deletion/relabeling). As discussed in [17], a vertex can be deleted or inserted only on the condition that it is isolated. Therefore, inserting or deleting a vertex over q^i results in $\lambda(q^i, q^{i+1}) = 1$. If a vertex v is relabeled, only the branch rooted at v will be affected. Hence, $\lambda(q^i, q^{i+1}) = 1$. In summary, $\lambda(q^i, q^{i+1}) = 1$ when we perform one vertex operation (inserting/deleting/relabeling a vertex) over q^i.

Above all, we have the following inequality:

$$\begin{aligned} \lambda(q, g) &\leq 1 \cdot k_1 + 1 \cdot k_2 \\ &\leq 1 \cdot (k_1 + k_2) \\ &\leq 1 \cdot mged(q, g). \end{aligned}$$

Thus, $mged(q, g) \geq \lambda(q, g)$. □

Thanks to the crucial nature that a single edit operation can affect only two branches, Theorem 4.1 guarantees that the mapping distance between $B(q)$ and $B(g)$ is a lower bound for the minimum graph edit distance.

Example 3. Consider the query graph q and two data graphs g_1 and g_2 in Fig. 1, where $\tau = 3$, $\lambda(q, g_1) = 3.5$, and $\lambda(q, g_2) = 3.5$. According to Theorem 4.1, we have $dist_B(q, g_1) = 3.5$ and $dist_B(q, g_2) = 3.5$, both of which are larger than τ. Therefore, g_1 and g_2 can be pruned safely.

A classical algorithm to compute the minimum weighted match is the Hungarian algorithm [34] with time complexity

$O(|V|^3)$, where $|V| = \max(|V(q)|, |V(g)|)$. Hence, we propose the compact branch filter that can be computed in the time complexity of $O(|V|\log|V|)$ in the next section.

4.2 Compact Branch Filter

The main cost of computing branch filter lies in finding the minimum weighted match between $B(q)$ and $B(g)$. However, we can reduce the complexity by introducing some constraints on the edge weights in the bipartite graph.

4.2.1 Definition of Compact Branch Filter

We define the compact branch distance of b_1 and b_2 in three cases. (1) if b_1 and b_2 are identical, their distance is 0. (2) if b_1 and b_2 have the same root label, but different sets of edge labels, their distance is $\frac{1}{2}$. The intuition is that one edge operation affects two branches. To avoid over-counting the number of edit operations, the compact distance is $\frac{1}{2}$ only if they are distinct just owing to having different edge label sets but the same root label. (3) if b_1 and b_2 have different root labels, their edit distance is 1, because it needs at least one operation over the root vertex to make the two branches identical.

Definition 4.4 (Compact Branch Distance). *Given two vertices v_1 and v_2, their branch structures are denoted as $b_1 = (l_1, ES_1)$ and $b_2 = (l_2, ES_2)$. The compact distance between b_1 and b_2 is defined as*

$$bed_C(b_1, b_2) = \begin{cases} 0, & l_1 = l_2 \wedge ES_1 = ES_2, \\ 1/2, & l_1 = l_2 \wedge ES_1 \neq ES_2, \\ 1, & l_1 \neq l_2. \end{cases}$$

Given two graphs q and g, we can also get two branch sets $B(q)$ and $B(g)$ (like the discussion in Section 4.1). Based on the minimum weighted match in the bipartite graph formed by $B(q)$ and $B(g)$ (an example is shown in Fig. 3), we can compute the compact distance between $B(q)$ and $B(g)$. Note that as opposed to Section 4.1, we use the compact branch distance (Definition 4.4) instead of branch distance (Definition 4.2). Definition 4.5 and Theorem 4.2 show the details of the new lower bound.

Definition 4.5. *Given two multisets of branch structures $B(q)$ and $B(g)$ with the same cardinality , and assume $P : B(q) \rightarrow B(g)$ is a bijection. The compact mapping distance between $B(q)$ and $B(g)$ is*

$$\lambda_C(q, g) = \min_P \sum_{b_i \in B(q)} bed_C(b_i, P(b_i)).$$

According to the compact mapping distance between $B(q)$ and $B(g)$, we can obtain a lower bound of edit distance, denoted as $dist_{CB}(q, g)$.

Theorem 4.2. *Given two graphs q and g, the minimum graph edit distance $mged(q, g)$ is no less than the compact mapping distance of their branch structures, i.e.,*

$$mged(q, g) \geq dist_{CB}(q, g) = \lambda_C(q, g)$$

.

Proof. Let $P = (p_1, p_2, \ldots, p_k)$ be an alignment transforming from q to g. Accordingly, there is sequence of graph $q = q^0 \rightarrow q^1 \rightarrow \cdots \rightarrow q^k = g$, where $q^i \rightarrow q^{i+1}$ indicates transforming q^i to q^{i+1} by operation p_i. Assume that there are k_1 edge insertion/deletion/relabeling operations, k_2 vertex insertion/deletion/relabeling operations in P, then $k_1 + k_2 = mged(q, g)$.

1) Edge insertion/deletion/relabeling. If an edge is inserted or deleted or relabeled over graph q^i, only two branches are affected. Thus, we can know that $\lambda_C(q^i, q^{i+1}) \leq 2 \cdot 0.5 = 1$ in the case of performing one edit operation of inserting or deleting or relabeling an edge over q^i.

2) Vertex insertion/deletion/relabeling. As discussed in [17], a vertex can be deleted only on the condition that it is an isolated vertex, and we can only insert an isolated vertex. If a vertex is inserted or deleted over q^i, $\lambda_C(q^i, q^{i+1}) = 1$. When the label of a vertex v is relabeled, only the branch rooted at v is affected. Hence, $\lambda_C(q^i, q^{i+1})$ is 1. In summary, $\lambda_C(q^i, q^{i+1}) = 1$ in the case of performing one edit operation of inserting or deleting or relabeling a vertex in q^i.

Thus, we have the following inequality: $\lambda_C(q, g) \leq k_1 + k_2 \leq mged(q, g)$, that is, $mged(q, g) \geq \lambda_C(q, g)$. □

Example 4. Considering the query graph q and two data graphs g_1 and g_2 in Fig. 1, we have $dist_{CB}(q, g_1) = 2$, because the solid lines in Fig. 3 form a mapping. Similarly, $dist_{CB}(q, g_2) = 2.5$.

More importantly, we propose a more efficient algorithm to compute the compact branch lower bound.

4.2.2 Computing the Bound of Compact Branch Filter

The computation of the compact branch lower bound consists of three major steps.

1) Remove the identical branch pairs (b_i, b_j) from $B(q)$ and $B(g)$, respectively, where $b_i \in B(q)$, $b_j \in B(g)$, and $b_i = b_j$. Each removal does not lead to increasing $dist_{CB}(q, g)$.
2) Remove the branch pairs (b_i, b_j) from $B(q)$ and $B(g)$, respectively, where $b_i \in B(q)$, $b_j \in B(g)$, the root vertices of b_i and b_j have the identical label. Assume that we remove Y branch pairs in this step. It increases the lower bound by $0.5 \times Y$, i.e., $dist_{CB}(q, g) = dist_{CB}(q, g) + 0.5 \times Y$.
3) Assume that there are Z branches left in $B(q)$ (including the blank branches, if any). We update the lower bound as $dist_{CB}(q, g) = dist_{CB}(q, g) + Z$.

In order to implement the steps above efficiently, we introduce *lexical branch order* for two branches. For ease of the presentation, let $l_i < l_j$ denote that (vertex or edge) label l_i is smaller than (vertex or edge) label l_j in the lexicographic order. ($l_i \leq l_j$ denotes that l_i is not larger than l_j in the lexicographic order).

Definition 4.6 (Lexical Branch Order). *Given two branches $b_1 = (l_1, ES_1)$ and $b_2 = (l_2, ES_2)$, $ES_1 = \{l_1, l_2, \ldots, l_m\}$ $(l_1 \leq l_2 \leq \cdots \leq l_m)$ and $ES_2 = \{l_1', l_2', \ldots, l_n'\}$ $(l_1' \leq l_2' \leq \cdots \leq l_n')$, we define that $b_1 \preceq b_2$ if 1) $l_1 < l_2$; or 2) $l_1 = l_2$ and $l_i \leq l_i'$ holds for the first different label pair $< l_i, l_i' >$*

from ES_1 and ES_2, where $l_i \in ES_1$ and $l_i' \in ES_2$, i.e., $l_j = l_j'$ for $1 \le j < i$ and $l_i \neq l_i'$.

First step. According to this lexical branch order over branches, we can sort the two multisets $B(q)$ and $B(g)$, i.e., $B(q) = \{b_1, b_2, \ldots, b_m\}$ ($b_1 \preceq b_2 \preceq \cdots \preceq b_m$), and $B(g) = \{b_1', b_2', \ldots, b_n'\}$ ($b_1' \preceq b_2' \preceq \cdots \preceq b_n'$). Thus, the identical branch pairs in $B(q)$ and $B(g)$ can be removed in a merge-sort way (lines 2-10 in Algorithm 2).

Second step. In this step, it is guaranteed that $B(q)$ and $B(g)$ have no identical branch pairs. Otherwise, they have been removed in the first step. According to Definition 4.4, a pair of branches (b_i, b_j') where $b_i \neq b_j'$ but with the same root vertex label, leads to 0.5 compact distance. Lines 2-20 in Algorithm 2 describe this process.

Third step. In this step, it is guaranteed that $B(q)$ and $B(g)$ have no branch pairs with the same root vertex label. Otherwise, they have been removed in the two previous steps. Assume that the number of branches left in $B(q)$ is Z. These Z left branches in q will lead to Z compact distance (line 21 in Algorithm 2).

Algorithm 2. CompactDistance($B(q)$, $B(g)$)

Input: Two sorted multisets of branch structures $B(q)$ and $B(g)$, where $B(q) = \{b_1, b_2, \ldots, b_m\}$ ($b_1 \preceq b_2 \cdots \preceq b_m$), $B(g) = \{b_1', b_2', \ldots, b_n'\}$ ($b_1' \preceq b_2' \preceq \cdots \preceq b_n'$)
Output: $dist_{CB}(q, g)$

```
1  dist_CB(q,g) ← 0
2  i ← 1, j ← 1
3  while i ≤ m && j ≤ n do
4      if b_i == b_j' then
5          remove b_i and b_j' from B(q) and B(g), respectively
6          i ← i + 1, j ← j + 1
7      else if b_i ≼ b_j' then
8          i ← i + 1
9      else
10         j ← j + 1
11 i ← 1, j ← 1
12 while i ≤ m && j ≤ n do
13     if l_i == l_j then
14         remove b_i and b_j' from B(q) and B(g), respectively
15         dist_CB(q,g) ← dist_CB(q,g) + 0.5
16         i ← i + 1, j ← j + 1
17     else if b_i ≼ b_j' then
18         i ← i + 1
19     else
20         j ← j + 1
21 Z ← the number of branches left in B(q)
22 dist_CB(q,g) ← dist_CB(q,g) + Z
23 return dist_CB(q,g)
```

Theorem 4.3. *Given two graphs q and g, Algorithm 2 gives their compact branch distance, i.e., $dist_{CB}(q, g)$.*

Proof. Assume that any other mapping generates X' identical branch pairs, Y' branch pairs satisfying the second step, and Z' branch pairs in the third step, the compact distance is $(0.5 * Y' + Z')$. According to Algorithm 2, it is obvious that $X \ge X'$ and $Z \le Z'$. (1) When $Y \le Y'$, $(0.5 * Y + Z) \le (0.5 * Y' + Z')$. (2) When $Y > Y'$, $H = (0.5 * Y' + Z') - (0.5 * Y + Z) = 0.5 * (Y' - Y) + (Z' - Z) \ge 0.5 * [(Y'P + Z') - (Y + Z)]$. Since $X + Y + Z = X' + Y' + Z'$ and $X \ge X', Y' + Z' \ge Y + Z$. Hence, $H \ge 0$. That is also to say $(0.5 * Y + Z) \le (0.5 * Y' + Z')$. □

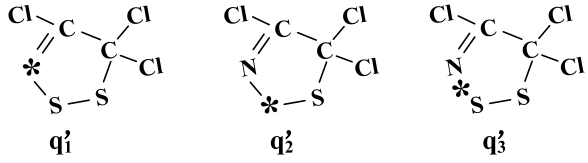


Fig. 4. A graph set of replacing results.

Complexity analysis. The time complexity of computing compact branch distance (Algorithm 2) is $O(|V|\log|V|)$, where $|V| = MAX\{|V(q)|, |V(g)|\}$. As the first two steps (lines 2-10 and lines 2-20) adopt the merge-sort style algorithm, it is easy to know the complexity is $O(max\{|V(q)|,|V(g)|\})$. The final step (lines 21-22) is a simple counting process, which also has the linear complexity. However, the complexity of sorting $B(q)$ and $B(g)$ is $O(|V|\log|V|)$.

As discussed above, the computation of compact distance is more efficient. However, when $\tau \ge 3$, neither g_1 nor g_2 can be pruned by the filter. Thus, we design another more powerful filter, called *hybrid filter* that incorporates both partitioning and branch strategies.

5 A Hybrid Filter

The previous sections present two different lower bounds $dist_P(q, g)$ and $\lambda(q, g)$ that are independent of each other. In this section, based on the ideas of the two lower bounds, we propose a new hybrid lower bound $dist_H(q, g)$ and prove that $dist_H(q, g) \ge MAX(dist_P(q, g), \lambda(q, g))$ (see Theorem 5.2). In other words, $dist_H(q, g)$ provides better (at least not lower) pruning power.

5.1 Pruning Strategy

We incorporate the partition-based and branch-based strategies together to enhance the pruning power. For a mismatching structure q_{s_i}, it requires at least one edit operation over q_{s_i} to match some substructure in g. We enumerate all possible one-step edits over q_{s_i} by replacing a vertex or an edge q_{s_i} with a wildcard.

Example 5. Consider query graph q and data graph g_1 in Fig. 1. Regarding the mismatching structure q_{s_1} (N−S) in Fig. 2a, there are three possible replacements $q^* = \{q_1', q_2', q_3'\}$, as shown in Fig. 4, where asterisk "*" represents a wildcard.

Let us consider q_1'. Similar to computing the branch-based lower bound in Section 4, we enumerate all branches in q_1' and g_1, and then build a bipartite graph between $B(q_1')$ and $B(g_1)$, as shown in Fig. 5. Different from branch edit distance in Definition 4.2, we re-define the distance between branches with wildcards in Definition 5.1. Obviously, we can obtain $\lambda(q_1', g_1)$ by finding the minimum weighted match in the bipartite graph.

Definition 5.1 (Wildcard-Branch Edit Distance). *Assume that we have two branches $b_1 = (l_1, ES_1)$ and $b_2 = (l_2, ES_2)$, where b_1 contains one wildcard on l_i or ES_1. The wildcard-branch edit distance between b_1 and b_2 is defined as follows:*

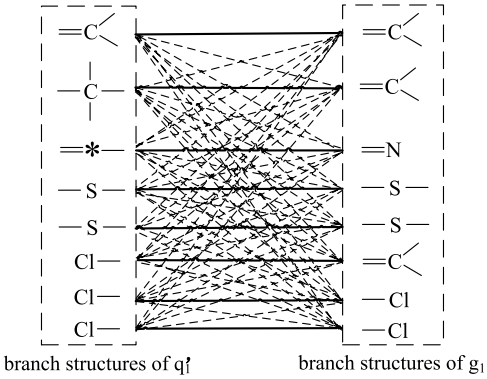


Fig. 5. Branch structures of q_1' and g_1.

$$bed(b_1, b_2) = T(l_1, l_2) + \frac{\Gamma(ES_1, ES_2)}{2} \tag{5}$$

$$\Gamma(ES_1, ES_2) = argmin_f \sum_{l \in ES_1} T(l, f(l)) \tag{6}$$

$$T(l_1, l_2) = \begin{cases} 0, & \text{if } \ l_1 = l_2 \ \ or \ \ l_1 = *, \\ 1, & otherwise, \end{cases} \tag{7}$$

where $f(\cdot)$ is a bijective function from ES_1 to ES_2.

Given a query graph q and a data graph g, let $MS(q, g)$ denote all non-overlapping mismatching structures of q with regard to g. For each mismatching structure, we generate 1-step replacement using wildcards. Let q^* denote all possible replacements. Considering all possible replacements, we use $\lambda(q^*, g_1) = MIN(\lambda_{q' \in q^*}(q', g_1))$.

Definition 5.2 (Hybrid Lower Bound). *Given a query graph q and a data graph g, the hybrid lower bound $dist_H(q, g)$ is defined as follows:*

$$dist_H(q, g) = |MS(q, g)| + \lambda(q^*, g),$$

where q^* denotes all possible replacements and $|MS(q, g)|$ is the number of non-overlapping mismatching structures of q with regard to g.

Theorem 5.1. $dist_H(q, g) \leq mged(q, g)$.

Proof. 1) For simplicity, we first assume that $|MS(q, g)| = 1$. Suppose that the optimal transforming sequence of graph is $q = q^0 \rightarrow q^1 \rightarrow \cdots \rightarrow q^k = g$. Since q_s is a mismatching substructure, there must be at least one edit operation over q_s. We assume that the edit operation happens at the state from q^i to q^{i+1} in the transforming sequence. Since q_s has no overlaps with $(q - q_s)$, we can move the edit operation (i.e., from q^i to q^{i+1}) to the beginning of the transforming sequence. Thus, we get another optimal transforming sequence $q = q^{0\prime} \rightarrow q^{1\prime} \rightarrow \cdots \rightarrow q^{k\prime} = g$, where the edit operation from $q^{0\prime} \rightarrow q^{1\prime}$ is over q_s. There are three possible edits over q_s, i.e., deleting an isolated vertex with label, substituting a/an vertex/edge label and deleting an edge. Thus, $q^{1\prime}$ is the graph after one-step edit over q_s. According to the edit distance definition, we know that $mged(q, g) = 1 + mged(q^{1\prime}, g)$.

Let q^* denote all possible one-step replacements over q_s. According to the distance between branches with wildcards (see Definition 5.1), it holds that $\lambda(q^{1\prime}, g) \geq \lambda(q^*, g)$. As we know that $mged(q^{1\prime}, g) \geq \lambda(q^{1\prime}, g)$ and $mged(q, g) = 1 + mged(q^{1\prime}, g)$, we conclude that $mged(q, g) \geq 1 + \lambda(q^*, g)$.

2) If $|MS(q, g)| > 1$, it means that there are $|MS(q, g)|$ non-overlapping mismatching substructures in q with regard to g. Since these mismatching substructures have no overlaps, we assume that the first $|MS(q, g)|$ edit operations in transforming sequence happen at the $|MS(q, g)|$ mismatching substructures, respectively; and each mismatching substructure has one-step edit. Suppose that $q^{|MS(q,g)|\prime}$ is the graph after one-step edit over these $|MS(q, g)|$ mismatching substructures. Similar to 1), we can prove that $mged(q, g) \geq |MS(q, g)| + \lambda(q^*, g)$. □

5.2 Tightness and Computation

The theoretical result of our hybrid lower bound is that it can provide higher (at least not lower) pruning power.

Theorem 5.2. *Given a query graph q and a data graph g, the following inequality holds.*

$$dist_H(q, g) \geq MAX(dist_P(q, g), \lambda(q, g)),$$

where $dist_H(q, g)$, $dist_P(q, g)$, and $\lambda(q, g)$ denote the hybrid, partition-based and branch-based lower bounds, respectively.

Proof. The proof contains two parts, i.e., $dist_H(q, g) \geq \lambda(q, g)$ and $dist_H(q, g) \geq dist_P(q, g)$.

1) Consider the optimal mapping P between q' ($q' \in q^*$) and g with mapping distance $\lambda(q', g)$. If we conduct $|MS(q, g)|$ operations over q' to get q and compute a new distance $\lambda'(q, g)$ using mapping P, we have $\lambda'(q, g) - \lambda(q', g) \leq |MS(q, g)|$ (because $|MS(q, g)|$ edits will produce $|MS(q, g)|$ distance at most). Since $\lambda(q, g)$ is the minimum mapping distance between q and g, we can obtain that $\lambda(q, g) \leq \lambda'(q, g)$. Thus, $\lambda(q, g) \leq \lambda(q', g) + |MS(q, g)|$.
2) Since the mapping distance $\lambda(q^*, g)$ is non-negative and $dist_H(q, g) = |MS(q, g)| + \lambda(q^*, g) = dist_P(q, g) + \lambda(q^*, g)$, $dist_P(q, g) \leq dist_H(q, g)$.

□

Example 6. Consider graphs q, g_1, and g_2 in Fig. 1. The hybrid lower bounds are $dist_H(q, g_1) = 4.5$ and $dist_H(q, g_2) = 5.5$, respectively. Recalling that the partition-based lower bounds are $dist_P(q, g_1) = 2$ and $dist_P(q, g_2) = 3$, respectively, and the branch-based lower bounds are $dist_B(q, g_1) = 3.5$ and $dist_B(q, g_2) = 3.5$, respectively. It is clear that the hybrid lower bounds are much tighter.

Given a set of disjoint mismatching structures (denoted as $MS(q, g)$) of query graph q with regard to g, we enumerate all possible replacements (denoted as q^*), and compute the mapping distance $\lambda(q^*, g) = MIN(\lambda_{q' \in q^*}(q', g))$. Finally, we derive the hybrid lower bound $dist_H(q, g) = |MS(q, g)| + \lambda(q^*, g)$. Algorithm 3 presents the detail.

Algorithm 3. HybridLB($B(q)$, $B(g)$, $MS(q,g)$)

Input: Two multisets of branch structures $B(q)$ and $B(g)$, and a set of mismatching structures $MS(q,g)$
Output: $dist_H(q,g)$
1 $q^* \leftarrow$ the set of replacing results
2 $\lambda(q^*,g) \leftarrow |V(q)| + |E(q)| + |V(g)| + |E(g)|$
3 **for** $q' \in q^*$ **do**
4 compute $\lambda(q',g)$
5 **if** $\lambda(q^*,g) < \lambda(q',g)$ **then**
6 $\lambda(q^*,g) = \lambda(q',g)$
7 $dist_H(q,g) \leftarrow dist_H(q,g) + |MS(q,g)|$
8 **return** $dist_H(q,g)$

Complexity analysis. Consider a mismatching structure $q_{s_i} \in MS(q,g)$. There are $|q_{s_i}|$ possible replacing results, where $|q_{s_i}|$ is the sum of vertex and edge numbers in q_{s_i}. Therefore, the cardinality of q^* is $|q^*| = \Pi_{q_{s_i} \in MS(q,g)} |q_{s_i}|$. The overall complexity is $O(|q^*| \cdot |V|^3)$.

The hybrid lower bound improves the pruning power. However, its pruning phase is not efficient. Therefore, we need to design an efficient pruning framework to guarantee both the effectiveness and efficiency of the pruning process.

Although the branch-based lower bound is not tighter than the hybrid lower bound, it is very efficient regarding the filtering phase, which is confirmed in our experiments (see Fig. 9). Thereupon, we can perform the branch-based filter first to obtain some candidates, and then conduct the hybrid filter over these candidates further. The pruning framework is denoted as *mixed filter*. The major benefit of mixed filter is that we can perform the hybrid pruning over a small set of data graphs rather than the whole dataset without influencing the pruning power. More discussions can be found in Section 7.3.

6 Index and Query Processing

6.1 U-Tree Index

Given a query q and a database D, we need to exhaustively compute the lower bounds of $mged(q,g)$ for all graphs g ($g \in D$) one by one. Obviously, this is a long and tedious process, especially when $|D|$ is very large. In order to avoid the sequential scan, we propose an index *u-tree* as follows.

Definition 6.1. *The statistics of one data graph g is denoted as $BNI(g) = \{B(g), \Sigma_V(g), \Sigma_E(g), VE(g), \quad VEP(g)\}$, where $B(g)$ denotes all branches in g, $\Sigma_V(g)$ and $\Sigma_E(g)$ denote all vertex and edge labels in g, respectively, $VE(g)$ and $VEP(g)$ denote all size-2 and size-3 structures in g, respectively.*

The union of $BNI(g_1)$ and $BNI(g_2)$ is the set union of five parts in $BNI(g_1)$ and $BNI(g_2)$.

Definition 6.2 (Union Operation "$\sqcup$"). $BNI(g_1) \sqcup BNI(g_2) = \{B(g_1) \cup B(g_2), \quad \Sigma_V(g_1) \cup \Sigma_V(g_2), \quad \Sigma_E(g_1) \cup \Sigma_V(g_2), \; VE(g_1) \cup VE(g_2), VEP(g_1) \cup VEP(g_2)\}$

According to the above definition, we can also recursively define the union of multiple $BNI(g_i)$, $i = 1, \ldots, n$.

Definition 6.3. *A* u-tree *is a height-balanced tree, where*

1) *Each leaf node stores $BNI(g)$ of a data graph g.*
2) *Each intermediate node N is the union of all its child nodes, i.e., $BNI(N) = BNI(g_1) \sqcup BNI(g_2) \sqcup \cdots \sqcup BNI(g_m)$, where $g_1, \cdots, g_m$ are the child nodes of N.*

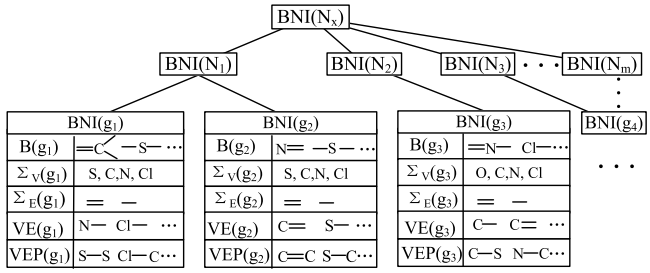


Fig. 6. *u-tree* Index.

Fig. 6 shows an example of *u-tree* index structure. The construction of *u-tree* is similar to *B-tree* and *R-tree*. We will discuss *u-tree* construction in Section 6.2. In this section, we assume that the *u-tree* has been built. We focus on how to utilize *u-tree* to improve the efficiency.

Considering an intermediate node N_i (in *u-tree*), we define the directed compact distance (denoted as $dist_{DCB}(q,N_i)$) and partition-based distance (denoted as $dist_{PL}(q,N_i)$) between q and N_i in Definitions 6.4 and 6.5, respectively. If one of the two distances is larger than τ, all its descendants of N_i can be pruned safely.

Definition 6.4. *The* directed compact distance *from query graph q to an intermediate node N_i, denoted as $dist_{DCB}(q,N_i)$, is the minimum compact edit distance of transforming $B(q)$ to $B(q)'$, such that $B(q)' \subseteq B(N_i)$.*

To compute the directed compact distance from q to N_i, we just need to modify Algorithm 2 slightly. We should subtract the common branches from $B(q)$ rather than the branch set with the maximum size.

Since the last step of our partition-based method may invoke expensive subgraph isomorphism verification, we only consider the first three steps for the intermediate nodes.

Definition 6.5. *The number of mismatching structures generated by the first three steps is defined as the* partition distance *between q and N_i, denoted as $dist_{PL}(q,N_i)$.*

With *u-tree* index, we have the following two theorems for the pruning.

Theorem 6.1. *If the compact distance $dist_{DCB}(q,N_i) \geq (\tau+1)$, all the child nodes of N_i can be pruned.*

Proof. Since N_i is the union of its child nodes, the common branches between q and N_i must be more than that between q and N_j, where N_j is one child node of N_i. Hence, we have $dist_{DCB}(q,N_j) \geq dist_{DCB}(q,N_i)$. If $dist_{DCB}(q,N_i) \geq (\tau+1)$, $dist_{DCB}(q,N_j) \geq (\tau+1)$. □

Theorem 6.2. *If $dist_{PL}(q,N_i) \geq (\tau+1)$, all the child nodes of N_i can be pruned.*

Proof. Since N_i is the union of its child nodes, the statistical information stored in node N_i is more than that in any of its child node N_j. The more statistical information are stored in N_i, the less mismatching structures will be obtained. Hence, $dist_{PL}(q,N_j) \geq dist_{PL}(q,N_i)$. If $dist_{PL}(q,N_i) \geq (\tau+1)$, $dist_{PL}(q,N_j) \geq (\tau+1)$. □

6.2 U-Tree Construction

Given a graph database D, we can build many different *u-trees* which may lead to different query performance. In

TABLE 2
Dataset Statistics

Dataset	$\|D\|$	$\|V\|$	$\|E\|$	$\|L_V\|$	$\|L_E\|$	max d
AIDS	42,687	45.7	47.71	4.37	2.06	4
PROTEIN	600	32.63	62.14	2.04	4.39	8
ER	100,000	64.86	157.07	9.39	43.53	17
SF	100,000	63.35	88.61	13.52	41.87	16

general, a good *u-tree* should provide high pruning ability for various queries. Considering a non-leaf node N_i, it is constructed by the union of its child nodes. Extremely, if all the child nodes are identical, $dist_{DCB}(q, N_i)$ is smallest, which favors a tight lower bound. It is intuitive that the smaller the difference of N_i's child nodes is, the tighter the lower bound $dist_{DCB}(q, N_i)$ will be. Hence, we present the following cost model, where g_j and g_k are two child nodes of N_i, and $|N_i|$ is the number of its child nodes.

$$\arg\min \sum_{N_i} \frac{\sum dist_{DCB}(g_j, g_k)}{|N_i|(|N_i|-1)}.$$

Since *u-tree* is analogue to *R-tree*, we can build *u-tree* by inserting the graphs sequentially. An insertion operation begins at the root and iteratively chooses a child node until it reaches a leaf node. We omit more details about the *u-tree* construction, since it is similar to *R-tree*.

6.3 Query Processing

Given a query graph q, we traverse the index *u-tree* starting from the root. Assume the current intermediate node is N_i, we first compute the directed compact distance $dist_{DCB}(q,\ N_i)$. If $dist_{DCB}(q, N_i) \geq (\tau + 1)$, we can safely prune the subtree rooted at node N_i. Otherwise, we compute $dist_{PL}(q, N_i)$. Analogously, if $dist_{PL}(q, N_i) \geq (\tau + 1)$, the subtree rooted at node N_i can be pruned. If neither $dist_{DCB}(q, N_i)$ nor $dist_{PL}(q, N_i)$ is larger than τ, the subtree will be accessed. Furthermore, if the current node is a leaf node g, we need to compute the compact branch lower bound $dist_{CB}(q, g)$ and the hybrid lower bound $dist_H(q, g)$.

Regarding those data graphs passing all the filters, we need to compute the minimum graph edit distances. Any existing methods [27], [35] can be employed.

7 Experimental Study

We evaluate the performance of our proposed method, and compare it with *c-star* [17], *k-AT* [31], *path-gram* [27], and *pars* [36] over both real and synthetic datasets.

7.1 Datasets and Settings

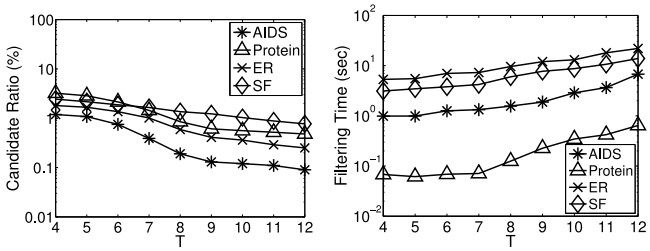
Real datasets. 1) AIDS is an antivirus screen compound dataset from the Developmental Theroapeutics Program in NCI/NIH.[4]

2) PROTEIN is a protein database from the Protein Data Bank,[5] where vertices represent secondary structure elements and are labeled with the corresponding enzyme class labels, edges indicate that two elements are neighbors.

Synthetic datasets. Two different synthetic graph models are used in our experiments, namely, *E*rdos *R*enyi (ER) and *S*cale *F*ree (SF). In ER model, N vertices are connected by M randomly chosen edges. The vertex degrees of SF graphs satisfy the power law distribution. We use the graph generator *gengraph win*[6] to generate SF graphs.

The statistics of the datasets are listed in Table 2, where the numerics from the third column to sixth column are average statistics, and d is the maximum vertex degree. We randomly select 100 graphs from each dataset as its query graphs, and average the query response time.

In this paper, all experiments are conducted on a P4 3.0 GHz machine with 4 G RAM running Linux. All programs are implemented in C++. The length of grams in *k-AT* and *path-gram* are set to be 1 and 3, which are the suggested parameter values in [27].



(a) *canRatio vs. T* (b) Filtering Time *vs. T*

Fig. 7. Effect of T for the disjoint-partition method.

7.2 Effect of T

As discussed in Section 3.3, the fourth step in the partition process needs to invoke a state-expansion based subgraph isomorphism verification algorithm (such as VF2 [11] and QuickSI [32]). To avoid consuming much time in this step, we restrict the state size no larger than T. So we first evaluate the effect of T by only performing the partition-based filter. We fix the dataset to be AIDS, Protein, ER(100 k), and SF (100 k), respectively, and set the threshold to be 3. We vary the parameter T from 4 to 12. Figs. 7a and 7b show the candidate ratio and filtering time *vs.* T. The candidate ratio (denoted as *canRatio*) is defined as Equation (8).

$$canRatio = \frac{candidate\ size}{|D|}, \tag{8}$$

where $|D|$ is the number of graphs in the database.

Fig. 7a shows that *candRatio* decreases as T increases. It indicates that the pruning ability increases with the growth of T. However, larger T leads to more filtering time, as presented in Fig. 7b, because the larger T will lead to larger search space. Fig. 7a shows that *canRatio* is stable when $T > 8$. Thus, we set $T = 8$ in our experiments.

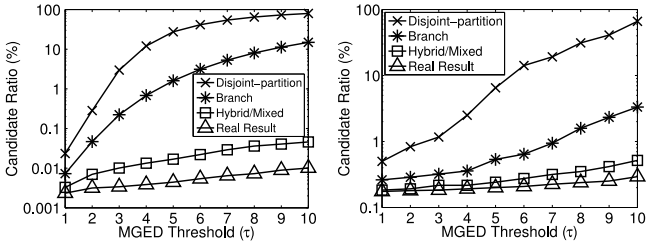
7.3 Evaluating Our Methods

We propose four filters, i.e., the disjoint-partition filter (or simply denoted by "**D**"), branch filter (or simply denoted

4. http://dtp.nci.nih.govdocsaidsaids_data.html

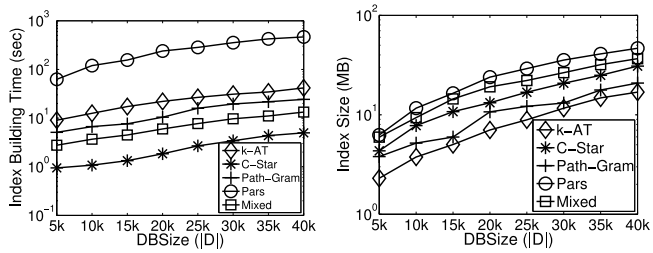
5. http://www.iam.unibe.chfkidatabasesiam-graph-databasedownload-the-iam-graph-database

6. http://fabien.viger.free.fr/liafa/generation/



(a) Results on AIDS (b) Results on Protein

Fig. 8. $canRatio\ vs.$ MGED threshold τ.



(a) Index Building Time (b) Index Size

Fig. 10. Offline performance on AIDS.

by "**B**"), hybrid filter (or simply denoted by "**H**"), and mixed filter (or simply denoted by "**M**").

Fig. 8 shows the pruning power of these methods by varying the MGED threshold τ from 1 to 10. The candidate ratios returned by all methods increase with the growth of τ. Notice that the hybrid and mixed filters have the same pruning power, which has been stated in Section 5.2. It is obvious that the hybrid/mixed filter is more powerful than using either disjoint-partition or branch only. Hence, the verification time will be reduced significantly employing the hybrid filter or the mixed filter.

Fig. 9 shows the response time (i.e., the filtering time plus the verification time) consumed by all these filters. The branch filter is the most efficient in terms of the filtering time. In comparison, the disjoint-partition filter may invoke the expensive subgraph isomorphism verification. Thus, it is slower than the branch filter. Incorporating both partition and branch strategies together, the hybrid filter consumes the most filtering time. As discussed in Section 5.2, the mixed filter utilizes the branch filter first (the most efficient regarding the filtering time) to obtain some candidates, and then conducts the hybrid filter (the most powerful pruning power) over the candidates. Therefore, the mixed filter is the most efficient regarding the overall response time.

7.4 Comparing with Existing Methods

In this section, we compare our method, i.e., the mixed filter, with existing methods k-AT [31] , c-star [17], SEGOS [37], path-gram [27], and pars [36].

7.4.1 Evaluating Offline Performance

We vary the size of database to evaluate the time of building index and the storage cost as shown in Figs. 10a and 10b, respectively. Regarding the indexing time, pars partitions data graphs into subgraphs one by one. Hence, it consumes more time than others. Since the depth of tree is 1 for *k-AT*, it is just the star structure defined in c-star. However, c-star only needs to enumerate all the star structures in data graphs without any complex index. Thus, it is superior to all the other methods. Because the size of branch is smaller than that of star, and we assign each branch a unique id to reduce index space, the space cost of $mixed\ filter$ is competitive as shown in Fig. 10b.

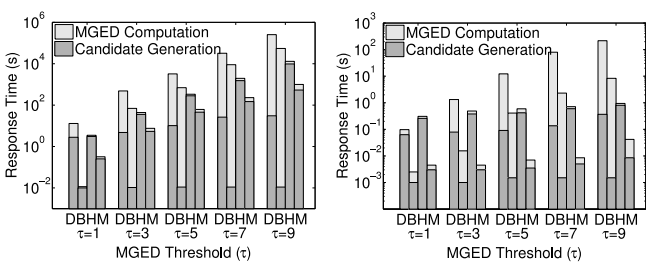
7.4.2 Evaluating Online Performance

We fix the datasets and vary the threshold τ from 1 to 10. Fig. 11 presents *canRatios* of different methods over $AIDS$, $Protein$. The less the candidate ratio is, the stronger the pruning ability will be. As shown in Fig. 11, canRatios generated by all these methods increase with the growth of τ. Fig. 11 shows that the candidate ratio of our proposed method (*mixed filter*) is the lowest, i.e., it has the strongest pruning power. What is more, the gap between the *mixed filter* and other methods gets larger with the growth of MGED threshold τ, which reveals that the pruning power of other methods is weaker compared with the *mixed filter*.

We also compare the efficiency of the *mixed filter* with other methods as shown in Fig. 12. Note that pars [36] outperforms c-star [17], SEGOS [37], and path-gram [27]. We just need to compare the mixed filter (denoted by "**M**") with k-AT (denoted by "**T**") and pars (denoted by "**P**"). Since the MGED computation is invoked during the verification phase, the verification time dominates the response time as shown in Figs. 12a and 12b. Although the filtering time of the mixed filter is not always the least, its overall response time is far less than that of its competitors. The main reason is that much less candidates are required to be verified benefitting from its powerful pruning ability.

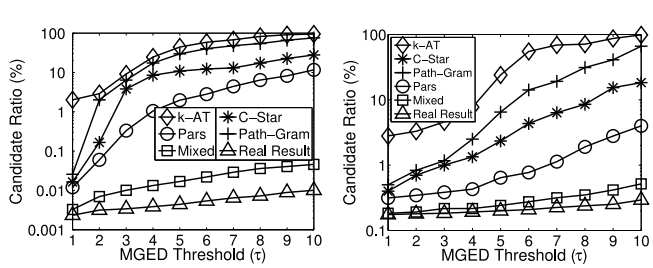
7.4.3 Evaluating Scalability

Effect of $|D|$. To study the effect of $|D|$, we vary the size of datasets, and set the MGED threshold to be 3. Fig. 13 presents the time efficiency of all these methods. As depicted in this



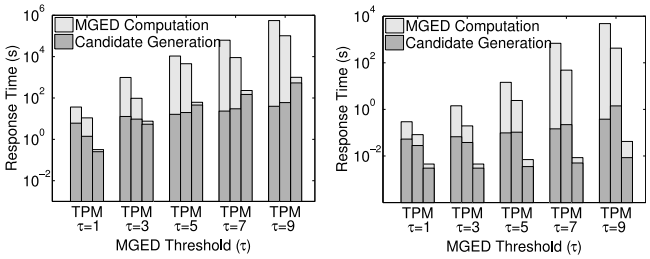
(a) Results on AIDS (b) Results on Protein

Fig. 9. Response time $vs.$ MGED threshold τ.



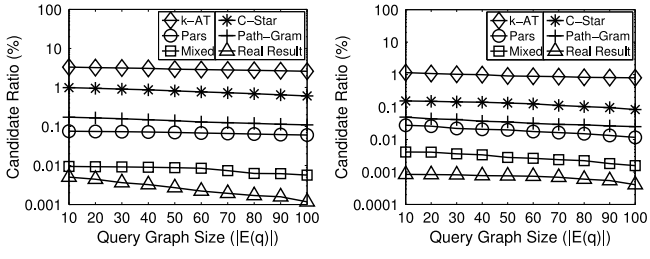
(a) Results on AIDS (b) Results on Protein

Fig. 11. $canRatio\ vs.$ MGED threshold τ.



(a) Results on AIDS (b) Results on Protein

Fig. 12. Filtering time $vs.$ MGED threshold τ.



(a) Results on ER (b) Results on SF

Fig. 14. CandRatio $vs.$ $|E(q)|$.

figure, the mixed filter outperforms k-AT by two orders and outperforms pars by one order of magnitude in terms of the overall response time, respectively. Comparatively, the k-At is rather inefficient since it suffers from the effect of large vertex degrees. Note that the time consumed by all these approaches is almost linear of the size of datasets, because the pruning abilities are stable for datasets of different size.

Effect of $|E(q)|$. In order to study the effect of the query graph size on the pruning ability, we fix the datasets (ER and SF) and randomly select groups of graphs of sizes from 10 to 100 from each dataset as query graphs. Since the real results get fewer with the growth of the size of query graphs, the numbers of candidates for all methods decrease with the increasing of $|E(q)|$ as shown in Fig. 14.

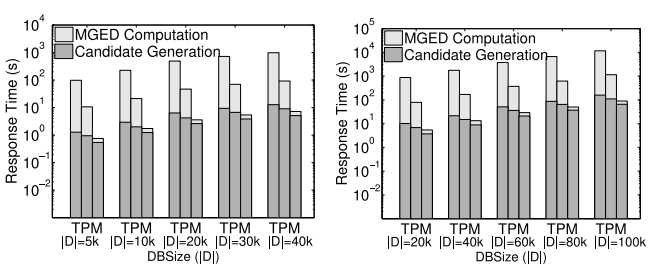
Effect of $|L_V|$. We also study the effect of the number of labels $|L_V|$ over ER and SF datasets. The results are presented in Fig. 15, where $|L_V|$ is increased from 5 to 50. It shows that if there are more labels in data graphs, the pruning abilities of all these approaches will be higher. This is because more information can be used in the computation of these lower bounds.

Effect of d. Fig. 16 shows the effect of maximum vertex degree on the pruning ability. It is obvious that the pruning abilities of k-AT, c-star, and path-gram decrease dramatically with the growth of the maximum vertex degree. It is because these methods are all based on the idea of n-gram, and one edit operation over the basic structure (tree, star, path) can affect $(d+1)$ structures at least. In contrast, the mixed filter utilizes branches, and one edit operation just affects two branches at most. Hence, the maximum vertex degree has little impact on the mixed filter.

8 Related Work

Lots of lower bounds are proposed to perform the pruning.

Number count filter [17]. The $mged(q,g)$ is not smaller than $dist_N(q,g) = ||V(g)| - |V(q)|| + ||E(g)| - |E(q)||$.



(a) Results on AIDS (b) Results on SF

Fig. 13. Filtering time $vs.$ $|D|$.

Label multiset filter [27]. Let $M_V(g)/M_E(g)$ be the multiset of vertex/edge labels. $dist_M(q,g) = \Gamma(M_V(q), M_V(g)) + \Gamma(M_E(q), M_E(g))$, where $\Gamma(X,Y) = max(|X|,|Y|) - |X \cap Y|$.

C-star [17]. A bipartite graph can be constructed with two sets of stars $S(q)$ and $S(g)$. Assume that the minimum weight matching in the bipartite graph is $\mu(q,g)$. The lower bound is computed according to Equation (9), where $\delta(q)$ and $\delta(g)$ are the maximum degrees in q and g, respectively.

$$dist_S(q,g) = \frac{\mu(q,g)}{max\{4, [max\{\delta(q),\delta(g)\}]+1\}}. \quad (9)$$

These star structures may have lots of overlapping structures, which results in a huge penalty by $max\{4, [max\{\delta(q),\delta(g)\}+1]\}$. Hence, the lower bound may be very small caused by the large denominator. Using star structures, SEGOS [37] introduces a two-level index and designs a novel search strategy. However, it follows the principle used in [17]. Thus, they have the same pruning power.

Tree-based n*-grams (k-AT [31]).* It defines an n-gram as a tree. If $mged(q,g) \leq \tau$, graphs q and g must share at least

$$dist_T(q,g) = \max(|V(q)| - \tau \cdot D_{tree}(q), |V(g)| - \tau \cdot D_{tree}(g))$$

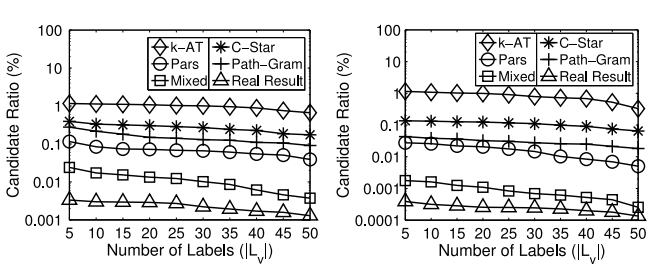
common n-grams, where D_{tree} is the maximum number of tree-based n-grams that can be affected by an edit operation.

Path-based n*-grams [27].* It defines an n-gram as a path. If $mged(q,g) \leq \tau$, graphs q and g must share at least

$$dist_A(q,g) = \max(|MG(q)| - \tau \cdot D_{path}(q), |MG(g)| - \tau \cdot D_{path}(g))$$

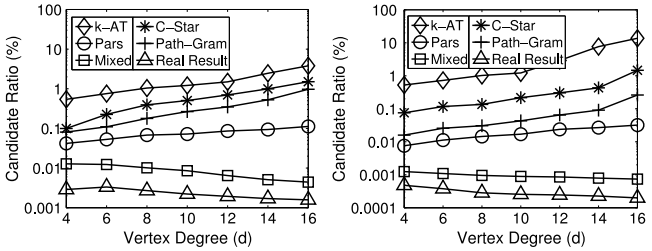
common n-grams, where $MG(q)$ and $MG(g)$ denote the multisets of n-grams in graphs q and g, D_{path} is the maximum number of path-based n-grams that can be affected by one edit operation. Clearly, if there is a high-degree vertex, the lower bound may be very small.

Recently, another partition-based method [36] is proposed. It divides data graphs into partitions offline and modifies the partitions according to the query graph online.



(a) Results on ER (b) Results on SF

Fig. 15. CandRatio $vs.$ $|L_V|$.



(a) Results on ER (b) Results on SF

Fig. 16. CandRatio $vs.$ maximum vertex degree d.

Although its partitioning strategy is distinct from ours, we can also incorporate it into our hybrid filter to obtain a tighter (at least not looser) lower bound.

In the verification phase, the most widely used method to compute exact graph edit distance is based on A^* algorithm incorporating heuristics [35]. It explores the space of all possible vertex mappings between two graphs to find the optimal mapping. Zhao et al. [27] improve A^* with two heuristics. Note that our work focuses on the filtering phase, i.e., the lower bound and the pruning strategy. They are independent of the verification algorithms.

To speed up the subgraph query processing, many feature-based indexing techniques, such as gIndex [12], Tree $+ \delta$ [38], and SING [39], have been proposed. The basic idea is that we can filter out data graph g if g does not contain the feature selected from query graph q. They focus on how to select features so as to enhance the pruning power. Distinct from the indexing techniques above, our partition-based pruning principle is counting the number of mismatching structures. Hence, we aim at finding as many mismatching structures as possible without caring about what the structures are.

9 Conclusions

In this paper, we study the graph similarity search under the graph edit distance constraint. Considering the limitations of existing approaches, we propose a systematic method for edit-distance based graph similarity search problem. Two lower bounds used to reduce the search space are proposed: one is based on the mismatching structures and the other one is based on branches. Importantly, we design a hybrid lower bound incorporating the both, and theoretically prove the hybrid lower bound is tighter (at least not looser) than the two lower bounds. In order to speed up the pruning process, an efficient pruning framework (mixed filter) is introduced. In addition, we carefully devise a uniform index, namely *u-tree*, to avoid explore the data graphs one by one. Extensive experiments over both real and synthetic datasets confirm that our proposed method outperforms the existing approaches significantly.

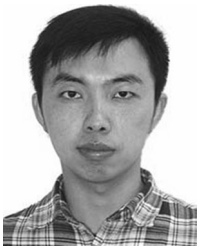
Acknowledgments

This work was supported by China 863 Project under Grant No. 2012AA011101, National Science Foundation of China (NSFC) under Grant No. 61370055 and 61272344, and CCF-Tencent Open Research Fund. L. Zou is the corresponding author.

References

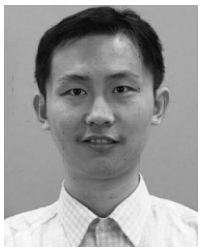
[1] D. Bonchev and D. H. Rouvray, *Chemical Graph Theory: Introduction and Fundamentals*. New York, NY, USA: Abacus Press, 1991.
[2] D. J. Watts, P. S. Dodds, and M. E. J. Newman, "Identity and search in social networks," *Science*, vol. 296, no. 5571, pp. 1302–1305, 2002.
[3] J. E. Beasley and N. Christofides, "Theory and methodology: Vehicle routing with a sparse feasibility graph," *Eur. J. Oper. Res.*, vol. 98, no. 3, pp. 499–511, 1997.
[4] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, "gStore: Answering SPARQL queries via subgraph matching," *Proc. VLDB Endowment*, vol. 4, no. 8, pp. 482–493, 2011.
[5] E. P. F. Chan and H. Lim, "Optimization and evaluation of shortest path queries," *VLDB J.*, vol. 16, no. 3, pp. 343–369, 2007.
[6] J. Cheng and J. X. Yu, "On-line exact shortest distance query processing," in *Proc. 12th Int. Conf. Extending Database Technol.: Adv. Database Technol.*, 2009, pp. 481–492.
[7] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM J. Comput.*, vol. 32, no. 5, pp. 1338–1355, 2003.
[8] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu, "Dual labeling: Answering graph reachability queries in constant time," in *Proc. 22nd Int. Conf. Data Eng.*, 2006, pp. 1–12.
[9] Y. Chen and Y. Chen, "An efficient algorithm for answering graph reachability queries," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 893–902.
[10] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.
[11] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004.
[12] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data.*, 2004, pp. 335–346.
[13] N. Przulj, "Geometric local structure in biological networks," in *Proc. IEEE Inf. Theory Workshop*, 2007, pp. 131–140.
[14] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognit. Lett.*, vol. 19, no. 3–4, pp. 255–259, 1998.
[15] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel, "Saga: A subgraph matching tool for biological graphs," *Bioinformatics*, vol. 23, no. 2, pp. 232–239, 2007.
[16] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 963–972.
[17] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.
[18] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood based fast graph search in large networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 901–912.
[19] S. Zhang, J. Yang, and W. Jin, "Sapper: Subgraph indexing and approximate matching in large graphs," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 1185–1194, 2010.
[20] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognit. Lett.*, vol. 18, no. 8, pp.–689–694, 1997.
[21] R. Ambauen, S. Fischer, and H. Bunke, "Graph edit distance with node splitting and merging, and its application to diatom idenfication," in *Proc. 4th IAPR int. Conf. Graph Based Representations Pattern Recognit.*, 2003, pp. 95–106.
[22] A. Robles-Kelly and E. R. Hancock, "Graph edit distance from spectral seriation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 3, pp. 365–378, Mar. 2005.
[23] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 18, no. 3, pp. 265–298, 2004.
[24] R. M. Marin, N. F. Aguirre, and E. E. Daza, "Graph theoretical similarity approach to compare molecular electrostatic potentials," *J. Chem. Inf. Model.*, vol. 48, pp. 933–944, 2008.
[25] S. EE, F. SH, and S. DA, "A network view of disease and compound screening," *Nat. Rev. Drug Discovery*, vol. 8, no. 4, pp. 286–295, 2009.
[26] G. K. Sims and L. E. Sommers, "Degradation of pyridine derivatives in soil," *J. Environ. Q.*, vol. 14, no. 4, pp. 580–584, 1985.

[27] X. Zhao, C. Xiao, X. Lin, and W. Wang, "Efficient graph similarity joins with edit distance constraints," in *Proc.IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 834–845.
[28] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 131–140.
[29] C. Xiao, W. Wang, and X. Lin, "Ed-join: An efficient algorithm for similarity joins with edit distance constraints," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 933–944, 2008.
[30] J. Feng, J. Wang, and G. Li, "Trie-join: A trie-based method for efficient string similarity joins," *VLDB J.*, vol. 21, no. 4, pp. 437–461, 2012.
[31] G. Wang, B. Wang, X. Yang, and G. Yu, "Efficiently indexing large sparse graphs for similarity search," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 3, pp. 440–451, Mar. 2012.
[32] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: An efficient algorithm for testing subgraph isomorphism," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 364–375, 2008.
[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[34] H. W. Kuhn, "The hungarian method for the assignment problem," in *Naval Res. Logistics*, vol. 2, pp. 83–97, 1955.
[35] K. Riesen, S. Fankhauser, and H. Bunke, "Speeding up graph edit distance computation with a bipartite heuristic," in *Proc. Mining Learn. Graphs*, 2007.
[36] X. Zhao, C. Xiao, X. Lin, Q. Liu, and W. Zhang, "A partition-based approach to structure similarity search," *Proc. VLDB Endowment*, vol. 7, no. 3, pp. 169–180, 2013.
[37] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin, "An efficient graph indexing method," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 210–221.
[38] P. Zhao, J. X. Yu, and P. S. Yu, "Graph indexing: Tree + delta >= graph." in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 938–949.
[39] R. D. Natale, A. Ferro, R. Giugno, M. Mongiovì, A. Pulvirenti, and D. Shasha, "Sing: Subgraph search in non-homogeneous graphs," *BMC Bioinformat.*, vol. 11, p. 96, 2010.



Weiguo Zheng received the BE degree from the School of Computer Science and Technology, China University of Mining and Technology (CUMT) in 2010. He is currently working toward the PhD degree at the Institute of Computer Science and Technology of Peking University, focusing on graph database management.

Lei Zou received the BS and PhD degrees in computer science from the Huazhong University of Science and Technology (HUST), in 2003 and 2009, respectively. Currently, he is an associate professor in the Institute of Computer Science and Technology of Peking University. His research interests include graph database and semantic data management.



Xiang Lian received the BS degree from the Department of Computer Science and Technology, Nanjing University, in 2003, and the PhD degree in computer science from the Hong Kong University of Science and Technology. He is currently an assistant professor in the Department of Computer Science at the University of Texas-Pan American. His research interests include probabilistic data management and probabilistic RDF graphs.

Dong Wang received the BS degree from the School of Electronics Engineering and Computer Science, Peking University in 2009. He is currently working toward the PhD degree at the Institute of Computer Science and Technology of Peking University, focusing on spatiotemporal query and keyword search over RDF.

Dongyan Zhao received the BS, MS, and PhD degrees from Peking University in 1991, 1994, and 2000, respectively. Currently, he is a professor in the Institute of Computer Science and Technology of Peking University. His research interests include information processing and knowledge management, including computer network, graph database, and intelligent agent.