2012 IEEE 28th International Conference on Data Engineering

Efficient Graph Similarity Joins with Edit Distance Constraints

Xiang Zhao[†] [§] Chuan Xiao[†] Xuemin Lin [*] [‡] [†] Wei Wang[†]

[†]*The University of New South Wales, Australia*
{xzhao, chuanx, lxue, weiw}@cse.unsw.edu.au
[‡] *East China Normal University, China*
[§] *NICTA, Australia*

***Abstract*—Graphs are widely used to model complicated data semantics in many applications in bioinformatics, chemistry, social networks, pattern recognition, etc. A recent trend is to tolerate noise arising from various sources, such as erroneous data entry, and find similarity matches. In this paper, we study the graph similarity join problem that returns pairs of graphs such that their edit distances are no larger than a threshold. Inspired by the q-gram idea for string similarity problem, our solution extracts paths from graphs as features for indexing. We establish a lower bound of common features to generate candidates. An efficient algorithm is proposed to exploit both matching and mismatching features to improve the filtering and verification on candidates. We demonstrate the proposed algorithm significantly outperforms existing approaches with extensive experiments on publicly available datasets.**

I. Introduction

Graphs have a wide range of applications and have been utilized to model complex data in biological and chemical information systems, multimedia, social networks, etc. There has been considerable interest in many fundamental problems in analyzing graph data. Various algorithms were devised to solve these problems, including frequent graph mining [32], [6], graph containment search and indexing [33], [12], [7], [38], etc.

Due to the existence of noisy and inconsistent data, a recent trend is to study similarity matches among graphs [34], [30], [27], [36], [24], [23], [28]. This body of work solves the problem of searching for graphs in a database that approximately contain or are contained by a query. Among the various graph similarity measures used in these studies, graph edit distance [4], [22] has been widely accepted for representing distances between graphs. Compared with alternative distances or similarity measures, graph edit distance has three advantages: (1) it allows changes in both vertices and edges; (2) it reflects the topological information of graphs; and (3) it is a metric and can be applied to any type of graphs. Due to these elegant properties, graph edit distance has been used in the context of classification and clustering tasks in various applications domains [1], [21]. However, the expensive computation of graph edit distance poses serious algorithmic challenges. In order to tackle the NP-hardness of the problem, a few algorithms have been proposed to either convert it to binary linear programming and compute the bounds [13] or seek unbounded suboptimal answers with heuristic techniques [9].

In this paper, we study the graph similarity join problem with graph edit distance constraints, a batch version of the graph similarity selection problem. It takes as input two sets of graphs, and returns pairs of graphs from each set such that their graph edit distances are no more than a given threshold.

There are several studies on the graph similarity selection problem with edit distance constraints; i.e., to find the graphs whose edit distances to the query are no larger than a threshold. These methods are either based on trees [28] or star structures [36]. The κ-AT algorithm proposed in [28] borrows the q-gram idea from the solution to string similarity problems [10], and defines a q-gram as a tree consisting of a vertex along with all those that can be reached in q-hops. A count filtering condition on common q-grams is established to qualify the candidate pairs that satisfy the graph edit distance constraint. However, it suffers from the looseness of the lower bound due to the huge impact of edit operations on common q-grams, and therefore is only effective against sparse graphs. The choice of q-gram length is also limited to a very small range, which usually consists of short q-grams, resulting in poor selectivity and thus large candidate size. The star structure proposed in [36] is exactly the same feature as the 1-gram defined by κ-AT. Unlike κ-AT, it computes the lower and upper bounds of graph edit distance with a bipartite matching between the star representations of two graphs. For graph similarity join problem, it has to invoke bipartite matching for every pair of graphs. The time complexity will be $O(|R||S||V|^3)$, where $|R|$ and $|S|$ are the dataset sizes, and $|V|$ is the number of vertices in a graph.

Distinct from the existing approaches, we explore a novel perspective of utilizing *path-based* q-grams. We find that the count filtering condition of path-based q-grams is tighter than using trees. This enables us to perform similarity join on denser graphs as well as choose longer q-grams for better selectivity. Another novelty is to exploit the valuable information provided by *mismatching* q-grams that cannot be matched in a candidate pair. Two filtering conditions are accordingly proposed so that the size of candidates can be substantially reduced. In addition, we elaborate how to

[*]Corresponding Author

speed up graph edit distance computation by further utilizing the two filtering conditions. The filtering and verification techniques constitute the GSimJoin algorithm. Its superior time efficiency to alternative methods is demonstrated through extensive experimental evaluation.

Our contributions can be summarized as follows:

- We solve the graph similarity join problem by introducing a new notion of q-grams based on paths. We develop the count filtering condition regarding the number of matching q-grams, which is tighter than using tree-based q-grams.
- We analyze mismatching q-grams and develop two filtering techniques to improve the performance of graph similarity join by optimizing both candidate generation and verification.
- We propose a new algorithm, GSimJoin, that integrates the proposed filtering and verification methods. We conduct extensive experiments using two publicly available datasets from different application domains. The proposed algorithm has been demonstrated to outperform other approaches.

The rest of the paper is organized as follows: Section II presents the problem definition and preliminaries. Section III introduces the definition of path-based q-gram on graphs and the corresponding count filter on matching q-grams. Sections IV and V present the two filtering techniques exploiting mismatching q-grams. Section VI elaborates the verification of candidates. Experimental results and analyses are presented in Section VII. Section VIII summarizes related work, and Section IX concludes the paper.

II. Preliminaries

A. Problem Definition

For ease of exposition, we focus on *simple* graphs in this paper. A simple graph is an undirected graph with neither self-loops nor multiple edges [1]. A labeled graph r can be represented as quadruple (V, E, l_V, l_E), where V is a set of vertices, and $E \subseteq V \times V$ is a set of edges. l_V and l_E are label functions that assign labels to vertices and edges, respectively. $V(r)$ denotes the vertex set of r, and $E(r)$ denotes the edge set. $|V(r)|$ and $|E(r)|$ represent the number of vertices and edges in r, respectively. $l_V(u)$ denotes the label of u, and $l_E(e(u,v))$ denotes the label of an edge between u and v, $u, v \in V$.

A graph r is isomorphic to another graph s if there exists a bijection $f : V(r) \rightarrow V(s)$ such that (1) $\forall u \in V(r)$, $f(u) \in V(s) \wedge l_V(u) = l_V(f(u))$, and (2) $\forall e(u,v) \in E(r)$, $e(f(u), f(v)) \in E(s) \wedge l_E(e(u,v)) = l_E(e(f(u), f(v)))$.

A graph edit operation is an edit operation to transform one graph to another [4], [22]. It can be one of the following six operations:

- Insert an isolated vertex into the graph.
- Delete an isolated vertex from the graph.
- Change the label of a vertex.
- Insert an edge between two disconnected vertices.

[1]Without loss of generality, our approach can be easily extended to directed graphs.

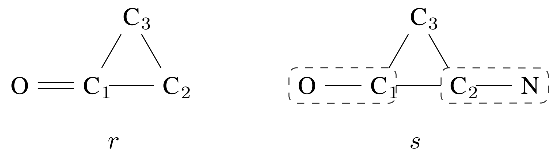


Fig. 1. Cyclopropanone and 2-Aminocyclopropanol

- Delete an edge from the graph.
- Change the label of an edge.

The graph edit distance between r and s, denoted by $ged(r, s)$, is the minimum number of edit operations that transform r to a graph isomorphic to s. It is shown that computing the graph edit distance between two graphs is NP-hard [36].

Example 1: Figure 1 shows the structure of cyclopropanone (r) and 2-aminocyclopropanol (s) molecules after omitting hydrogen atoms. They have been used in investigations of potential antiviral drugs [8]. For ease of illustration, we add subscripts to carbon atoms, while C_1, C_2 *and* C_3 *correspond to the same label in real data. Single and double lines indicate different chemical bonds, represented in edge labels in real data. The graph edit distance between* r *and* s *is 3.*

We formalize the graph similarity join problem as follows.

Given two sets of graphs R and S, a graph similarity join with edit distance threshold τ returns pairs of graphs from each set, such that their graph edit distance is no larger than τ; i.e., $\{\langle r, s\rangle \mid ged(r, s) \leq \tau, r \in R, s \in S\}$. Assuming there is a unique identifier recorded in id for each graph, this paper will focus on the self-join case; i.e., $\{\langle r_i, r_j\rangle \mid ged(r_i, r_j) \leq \tau \wedge r_i.id < r_j.id, r_i \in R, r_j \in R\}$.

B. Tree-based q-gram Approach

The string similarity problem with edit distance constraints has been extensively studied [10], [31], [14], [15], [35], [37], [29], [18]. Among them, many prevalent approaches are based on *q-grams* [10], [31], [14], [18], namely, substrings of length q. Since an edit operation will only affect a limited number of q-grams, similar strings will have certain amount of overlap between their q-gram sets [2]. Based on this observation, these approaches essentially relax the edit distance constraint to a weaker count constraint on the number of common q-grams called *count filtering*.

Inspired by the idea of q-gram on string similarity, [28] proposed κ-AT algorithm that defines the q-grams on graphs based on *trees*. For each vertex u, a tree-based q-gram is a set of vertices that can be reached from u in q hops, represented in a breadth-first-search tree rooted at u.

Example 2: Consider r *in Figure 1 and* $q = 1$. *There are four 1-grams of* r, *as shown in Figure 2. The first 1-gram appears twice.*

[2]q-grams in strings are accompanied by their starting positions in the string, and thus there is no duplicate.

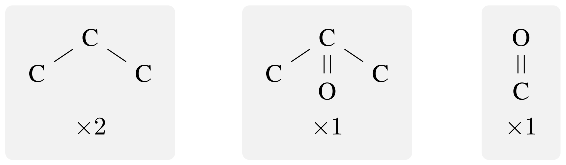


Fig. 2. Tree-based q-grams

The maximum number of q-grams that can be affected by an edit operation is shown as

$$D_{tree} = 1 + \gamma \cdot \frac{(\gamma - 1)^q - 1}{\gamma - 2},$$

where γ is the maximum vertex degree in the graph. κ-AT algorithm exploits the constraint that two graphs r and s must share at least a number of common q-grams if they are within graph edit distance τ:

$$LB_{tree} = \max(|V(r)| - \tau \cdot D_{tree}(r), |V(s)| - \tau \cdot D_{tree}(s)).$$

A pair of graphs that satisfies the lower bound test is called a candidate pair. Note that it does not necessarily satisfy the graph edit distance constraint. Therefore, graph edit distance calculation will be invoked for every candidate pair that survives this count filter.

κ-AT algorithm is associated with the drawback that the lower bound of common q-grams is usually loose. It may become equal to or even less than zero if there is a vertex with high degree in the graph, and we call such phenomenon *underflowing*. This problem results in the following dilemma: We have to use very short q-grams, e.g., 1-grams, to ensure the pairs of graphs to have at least one common q-gram so that the all-pair comparison due to underflowing can be avoided; however, short q-grams suffer from poor performance problems as they are usually frequent and hence yield large candidate size. Consider the two graphs in Figure 1 and $\tau = 1$, the lower bound is only 1 if we use 1-grams, and becomes less than zero when longer q-grams are applied.

III. A Path-based q-gram Method

Seeing the drawback of the tree-based q-gram approach, we seek a new way of defining q-grams on graphs. Since q-grams defined on strings are *sequences*, we may choose paths in a graph as its q-grams, as paths are convertible to "sequences". Next we formally introduce the definition of path-based q-grams on graphs.

A. Definition of Path-based q-gram

Definition 1 (path-based q-gram): A path-based q-gram in a graph r is a simple path of length q.

"Simple" means that there is no repeated vertex in the path. Since a path has two ends, namely, start vertex and end vertex, two sequences can be formed by concatenating the vertex and edge labels from both ends. We only keep the lexicographically smaller as a q-gram. Since the length of a path can be zero for the case of a single vertex, a 0-gram will be a single vertex. In the rest of the paper, we use "path-based q-gram" and "q-gram" interchangeably when there is no ambiguity.

Example 3: Consider the two graphs in Figure 1 and $q = 1$. There are four 1-grams in r:

$$\texttt{C=O}(\times 1)$$
$$\texttt{C-C}(\times 3)$$

and five 1-grams in s:

$$\texttt{C-N}(\times 1)$$
$$\texttt{C-O}(\times 1)$$
$$\texttt{C-C}(\times 3)$$

Before developing count filtering condition for path-based q-grams, we first study the effect of an edit operation on a graph's q-grams. Let Q_r denote the multiset of q-grams in r, and Q_r^u denote the multiset of q-grams that contain the vertex u. The following theorem shows how many q-grams in Q_r will be affected when an edit operation occurs in r.

Theorem 1: An edit operation on r will affect at most $(D_{path}(r) = \max_{u \in V(r)} |Q_r^u|)$ q-grams in Q_r.

Proof: We enumerate the effect of various edit operations:

- *Insert an isolated vertex into the graph.* No q-gram in Q_r will be affected.
- *Delete an isolated vertex from the graph.* The number of q-grams affected is either 1 when $q = 0$, or 0 otherwise.
- *Change the label of a vertex.* Suppose u's label is changed. This will affect $|Q_r^u| \le \max_{u \in V(r)} |Q_r^u|$ q-grams.
- *Insert an edge between two disconnected vertices.* No q-gram in Q_r will be affected.
- *Delete an edge from the graph.* Suppose $e(u, v)$ is deleted. The number of affected q-grams is $\max(|Q_r^u|, |Q_r^v|) \le \max_{u \in V(r)} |Q_r^u|$.
- *Change the label of an edge.* This will affect the same number of q-grams as deleting an edge from the graph.

■

According to Theorem 1, the count filtering condition for path-based q-grams can be established as:

Lemma 1 (Count Filtering): Consider two graphs r and s. If $ged(r, s) \le \tau$, r and s must share at least

$$LB_{path} = \max(|Q_r| - \tau \cdot D_{path}(r), |Q_s| - \tau \cdot D_{path}(s)) \tag{1}$$

common q-grams.

Example 4: Consider Figure 1, $\tau = 1$, and $q = 1$. Changing the label of $\texttt{C}_1$ gives the maximum $|Q_r^u| = 3$ for both graphs. Therefore the lower bound of common q-grams is $\max(4 - 3, 5 - 3) = 2$. Even when q is 2, the lower bound of common q-grams is still above zero, as given by $\max(5 - 5, 7 - 6) = 1$.

B. *Comparison with Tree-based q-grams*

Now we compare the effect of edit operations on tree-based and path-based q-grams.

- For $q = 1$, consider r in Figure 1. All the tree-based q-grams can be affected by an edit operation on $\texttt{C}_1$, while the path-based q-gram consisting of $\texttt{C}_2$ and $\texttt{C}_3$ will still be kept. This example showcases that path-based q-grams can preserve more common structural information than tree-based q-grams, excluding the affected part.
- For longer q-grams, the number of vertices covered by a tree-based q-gram increases *exponentially* with the length q. Any edit operation resident on these vertices will make this q-gram mismatched. On the contrary, the coverage of a path-based q-gram increases *linearly* with the length q, and therefore the probability being hit by an edit operation is much decreased.

Compared with tree-based q-grams, path-based q-grams have the advantage of presenting tighter lower bounds, and this will deliver the chance of using longer q-grams in seek of better selectivity and runtime performance.

C. *Prefix Filtering*

An efficient way to find the pairs of graphs that satisfy the count filtering condition is to use inverted index [2]. An inverted index maps each q-gram w to a list of identifiers of graphs that contain w. With the inverted index built for all the graphs in the data collection, we can scan each graph r, probe the index using every q-gram of r, and produce a set of candidates. Merging these candidates gives the actual intersection of q-grams and the graph pairs that meet the lower bound.

The main performance bottleneck in accessing inverted index is that the inverted lists of some q-grams can be very long. For example, the carbon chain $\texttt{C} - \texttt{C} - \texttt{C}$ exists in most chemical compounds in AIDS dataset. These long inverted lists will incur prohibitive overhead when accessed. In addition, a large number of candidate pairs will be produced if they share such q-grams. Existing approaches to string similarity problem address this bottleneck by employing prefix filtering technique [5], [31], [18] to quickly filter out the candidate pairs that are guaranteed not to meet the count filtering condition. The intuition is that if two multisets of q-grams meet the lower bound constraint, they must share at least one common q-gram if we look into part of the q-grams.

Figure 3 illustrates the idea of prefix filtering. Suppose the q-grams in two multisets are *sorted* in the same ordering. l is the total number of q-grams in both multisets. The unshaded cells are prefixes. If Q_r and Q_s have no common q-gram in their prefixes, the number of their common q-grams is no more than $LB_{path} - 1$. We formally state the prefix filtering principle for graph similarity joins in Lemma 2.

Lemma 2 (Prefix Filtering): Consider two graphs r, s, their corresponding q-gram multisets Q_r, Q_s, and a global ordering $\mathcal{O}$ of the q-gram universe. Let Q_r and Q_s be sorted in the order of $\mathcal{O}$, and the p-prefix be their first p elements. If $|Q_r \cap Q_s| \geq \alpha$, then the $(|Q_r| - \alpha + 1)$-prefix of Q_r and the $(|Q_s| - \alpha + 1)$-prefix of Q_s must have at least one common q-gram.

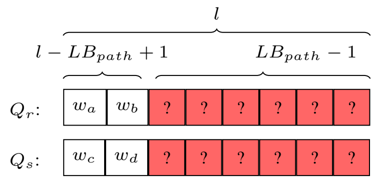


Fig. 3. Illustration of Prefix Filtering

In order to achieve a small candidate size and fast execution speed, rare q-grams are favored in prefixes. Therefore we sort the multiset of q-grams in each graph in ascending order of *document frequency*, the number of graphs that contain the q-gram.

D. *Graph Join Algorithm*

Combining count filtering for path-based q-grams and prefix filtering, we have the basic GSimJoin algorithm (Algorithm 1). The algorithm takes as input a collection of graphs, and follows an index nested loop join style, maintaining an in-memory inverted index on-the-fly. It iterates through each graph $r \in R$. For each q-gram w in Q_r's prefix, it probes the inverted index to find other graphs s that contain w in their prefixes. The candidates will be sent into Verify, and checked by (1) count filtering; and then (2) the expensive graph edit distance computation to tell if they are join results. According to Lemma 1 and 2, the prefix length is $\tau \cdot D_{path}(r) + 1$ for each graph r (Line 6). In addition, the numbers of vertices and edges in r and s must have the difference within τ. This *size filtering* is included in Line 9.

In the next two sections, we will study how to exploit the information provided by *mismatching* q-grams to gain further efficiency. Although the similar property also happens for strings and has been investigated in [31], the scenario on graphs is much more challenging. First, the q-grams on strings have starting positions, and hence are easy to locate, while the q-grams on graphs do not have such attribute. Second, the minimum edit operation problem on strings is of polynomial time complexity while it is NP-hard on graphs. We will propose two non-trivial techniques on graphs to reduce both index and candidate sizes.

IV. Minimum Edit Filtering

We first show an illustrative example.

Example 5: Consider Figure 1, $\tau = 1$, and $q = 1$. The count filtering lower bound is 2, while the two graphs share 3 q-grams (see Example 3) and therefore will survive count filtering. However, if we consider the two mismatching q-grams in s: $\texttt{C-O}$ *and* $\texttt{C-N}$*, it can be seen that they are disjoint (see the two bounded regions in s). It takes at least two edit operations to affect them. Obviously, we can infer a lower bound of the graph edit distance between r and s to be 2 and hence prune this pair. This motivates us to find the minimum number of edit operations that can cause the observed mismatching q-grams.*

Algorithm 1: GSimJoin (R, τ)

```
Input  : $R$ is a collection of graphs; $\tau$ is a graph edit
         distance threshold; $\mathcal{O}$ is a global ordering of $q$-grams.
Output : $S = \{ \langle r, s \rangle \mid ged(r, s) \leq \tau \}$.
1  $S \leftarrow \emptyset$;
2  $I_i \leftarrow \emptyset$ $(1 \leq i \leq |U|)$ ;            /* inverted index */
3  for each $r \in R$ do
4      $A \leftarrow$ empty map from id to boolean;
5      $Q_r \leftarrow r$'s $q$-grams sorted in $\mathcal{O}$;
6      $p_r \leftarrow \tau \cdot D_{path}(r) + 1$;
7      for $i = 1$ to $p_r$ do
8          $w \leftarrow Q_r[i]$;
9          for each $s \in I_w$ such
           that $\mathrm{abs}(|V(r)| - |V(s)|) + \mathrm{abs}(|E(r)| - |E(s)| \leq \tau$
           and $A[s]$ has not been initialized do
10             $A[s] \leftarrow$ true ;      /* find a candidate */
11         $I_w \leftarrow I_w \cup \{ r \}$ ;   /* index for $q$-gram $w$ */
12     $S \leftarrow S \cup$ Verify($r, A$);
13 return $S$
```

The above example evidences the implication of edit operations occur on disjoint q-grams and the existence of redundancy within prefixes. Since we choose increasing document frequency as the global ordering on q-gram multisets, the rarest q-grams reside in the beginning of prefixes, while the end of prefixes are relatively frequent q-grams. Both index size and the candidates passing prefix filtering can be reduced if we are able to remove the redundancy and avoid frequent q-grams with shortened prefixes.

A. Minimum Graph Edit Operations

Example 5 illustrates the case where mismatching q-grams are disjoint. To handle the general case where q-grams may overlap, we formulate the *minimum graph edit operation* problem: Given a multiset of q-grams Q, find the minimum number of graph edit operations that can affect all the q-grams in Q.

Theorem 2: The minimum graph edit operation problem is NP-hard.

Proof: (sketch) It can be shown the q-grams affected by all the other edit operations are a subset of the q-grams affected by changing vertex label. The minimum graph edit operation problem can be reduced from the set cover problem by treating q-grams as elements and vertices as sets. Therefore the minimum graph edit operation problem is NP-hard. ■

Despite its NP-hardness, the problem can be solved with an exact algorithm enumerating the positions of edit operations, since we only concern whether the answer is within or beyond τ. The time complexity is $O(|V|^\tau + |Q|)$, where $|V|$ is the number of vertices contained by the q-grams in Q. To alleviate the problem of large $|V|$, we may compute an approximate answer using the greedy algorithm [3] with an approximation ratio of $\ln |Q| - \ln \ln |Q| + 0.78$ [25]. The time complexity is reduced to $O(\tau(|V| + |Q|) \log |Q|)$.

[3]The greedy algorithm for set cover problem chooses the set which contains the largest number of uncovered elements at each stage.

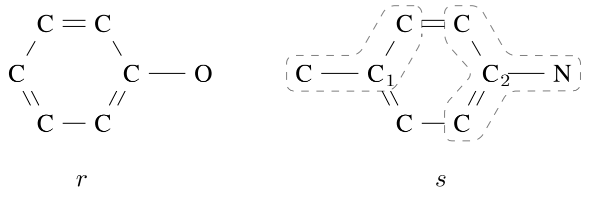


Fig. 4. Example of Minimum Edit Operation

Algorithm 2 presents the approximate algorithm and it is guaranteed to return a lower bound of the exact answer.

Algorithm 2: MinEditLowerBound (Q)

```
Input  : $Q$ is a multiset of $q$-grams.
Output : A lower bound of the minimum
         edit operations that affect all the $q$-grams in $Q$.
1 edit $\leftarrow$ compute $min - edit(Q)$ with the greedy algorithm;
2 return $\frac{\text{edit}}{\ln |Q| - \ln \ln |Q| + 0.78}$
```

Algorithm 3: MinEdit (Q)

```
Input  : $Q$ is a multiset of $q$-grams.
Output : The exact minimum
         edit operations that affect all the $q$-grams in $Q$.
1 edit $\leftarrow$ compute exact $min - edit(Q)$;
2 return edit
```

Example 6: Figure 4 shows the structure of phenol (r) and toluidine (s) molecules. Suppose q is 2. There are three mismatching q-grams from s to r: `C-C-C`, `C-C-N`, *and* `C=C-N`, *as bounded in the figure. At least two minimum edit operations are needed to make these three q-grams mismatch, e.g., changing the vertex label of* $\mathtt{C}_1$ *and* $\mathtt{C}_2$.

It is noteworthy to mention the following two properties, which are essential to the filtering techniques we are going to present in the rest of the paper. Let *min-edit*(Q) denote the minimum graph edit operations for a multiset of q-grams Q.

Proposition 1 (Monotonicity):

$$\textit{min-edit}(Q) \leq \textit{min-edit}(Q') \leq ged(r, s), \forall Q \subseteq Q' \subseteq Q_r \backslash Q_s.$$

Proposition 2 (Disconnectivity): $\textit{min-edit}(Q_1 \cup Q_2) = \textit{min-edit}(Q_1) + \textit{min-edit}(Q_2)$, if $\forall q_i \in Q_1, q_j \in Q_2, q_i \cap q_j = \emptyset$.

B. Minimum Prefix Length

Recall Example 5, although the lower bound of common q-grams is LB_{path}, it is likely that the minimum edit operations that occur on mismatching q-grams have already exceeded τ, and thus the candidate pair should be discarded. Based on this assumption, our task becomes seeking a *minimum* prefix such that at least $\tau + 1$ edit operations are needed to affect all the q-grams in the prefix. In this case, r and s will be guaranteed not to meet the graph edit distance constraint if all the q-grams in their prefixes are mismatched.

Algorithm 4: MinPrefixLen (Q_r)

```
Input  : $Q_r$ is a sorted multiset of $q$-grams of graph $r$.
Output : The minimum prefix length of $Q$.
1  left $\leftarrow \tau + 1$;    right $\leftarrow \tau \cdot D_{path}(r) + 1$;
2  while left < right do
3      mid $\leftarrow$ (left + right)/2;
4      edit $\leftarrow$ MinEditLowerBound($Q_r[1\,..\,\text{mid}]$);
5      if edit $\leq \tau$ then left $\leftarrow$ mid + 1;
6      else right $\leftarrow$ mid;
7  right $\leftarrow$ left; left $\leftarrow \tau + 1$;
8  while left < right do
9      mid $\leftarrow$ (left + right)/2;
10     edit $\leftarrow$ MinEdit($Q_r[1\,..\,\text{mid}]$);
11     if edit $\leq \tau$ then left $\leftarrow$ mid + 1;
12     else right $\leftarrow$ mid;
13 return left
```

The monotonicity (Proposition 1) enables us to find the minimum prefix length for a multiset of q-grams Q_r with a binary search within the range of $[\tau+1, \tau \cdot D_{path}(r)+1]$, as presented in Algorithm 4. It performs two rounds of binary search. In the first round, the greedy algorithm is called to find the lower bound of the answer to minimum graph edit operation problem. The result is used as the upper bound of the second round binary search, in which the exact algorithm is applied.

Lemma 3 (Minimum Edit Filtering): Denote the minimum prefix length for the q-grams of r and s as p_r and p_s, respectively. If $ged(r,s) \leq \tau$, Q_r's p_r-prefix and Q_s's p_s-prefix must have at least one common q-gram.

Lemma 3 states the minimum edit filtering. To apply this filtering in the join algorithm, we replace Line 6 in Algorithm 1 with "$p_r \leftarrow$ MinPrefixLen(Q_r)".

Example 7: Consider s in Figure 1 and its five 1-grams sorted according to the order they are listed in Example 3. When τ is 1, the minimum prefix length is 2, while the prefix length before using minimum edit filtering is 4.

V. Label Filtering

In this section, we introduce another approach of exploiting the labels in mismatching q-grams.

A. Exploiting the Labels in Mismatching q-grams

Although the minimum edit filtering can estimate a lower bound of graph edit distance, it works in a pessimistic way assuming the edit operations are *scattered*. However, it is likely that several edit operations are clustered and incurred by the same mismatching q-gram.

Example 8: Consider Figure 1, $\tau = 1$, and $q = 1$. The two mismatching q-grams are bounded in dashed lines. If we compare the labels in the mismatching q-gram in the right bounding box with those in r, they already incur at least one edit operation because there is no nitrogen atom (N) in r.

Motivated by this idea, we are able to establish a lower bound of graph edit distance from the labels in mismatching q-grams. Denoting $L_V(r)$ the multiset of the vertex labels in

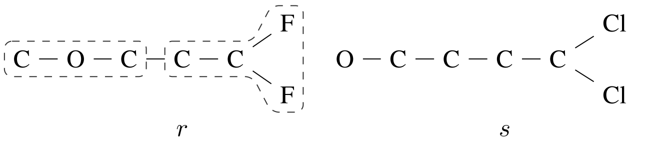


Fig. 5. Example of Local Label Filtering

r, and $L_E(r)$ the multiset of the edge labels in r, we state the local label filtering for graph edit distance:

Lemma 4 (Local Label Filtering): If $ged(r,s) \leq \tau$, $|L_V(r')\backslash L_V(s)| + |L_E(r')\backslash L_E(s)| \leq \tau$ for any subgraph r' of r.

Applying local label filtering on whole graphs immediately yields global label filtering.

Lemma 5 (Global Label Filtering): If $ged(r,s) \leq \tau$, $\Gamma(L_V(r), L_V(s)) + \Gamma(L_E(r), L_E(s)) \leq \tau$, where $\Gamma(A,B) = \max(|A|,|B|) - |A \cap B|$.

B. Implementation of Local Label Filtering

Although the local label filtering can be applied on any subgraphs, we choose as a heuristic to use it on the subgraphs containing at least one mismatching q-gram, since mismatching q-grams may imply difference in vertex and edge labels. In addition, we employ minimum edit filtering to enhance the power of local label filtering. Recall the disconnectivity of minimum graph edit operations (Proposition 2), the observed mismatching q-grams can be articulated and form a set of connected components. We may derive the lower bound of graph edit distance in the whole graph by computing that in each component and summing them up.

Algorithm 5: LocalLabelFilter(Q, s)

```
Input  : $Q$ is a multiset of mismatching $q$-grams from $r$ to $s$.
Output : A lower bound of $ged(r,s)$.
1  $C \leftarrow$ the connected components formed by $Q$;
2  total $\leftarrow 0$;
3  for each $c_i \in C$ do
4      edit-loc $\leftarrow$ MinEdit($c_i$);
5      edit-con $\leftarrow |L_V(c_i)\backslash L_V(s)| + |L_E(c_i)\backslash L_E(s)|$;
6      total $\leftarrow$ total + max(edit-loc, edit-con);
7  return total
```

Algorithm 5 explains the implementation of the enhanced local label filtering after including minimum edit filtering. For each connected component consisting of one or more mismatching q-grams, we compute the minimum edit operations that can result in these mismatching q-grams using (1) minimum edit filtering; and (2) local label filtering. The larger one is then chosen as the ged lower bound within this component, and added up to the total ged lower bound. The time complexity of the algorithm is $O(|V|^{\tau}+|Q|+|E|)$. In case of large V, we may calculate an approximate answer to the minimum edit operation problem with the greedy algorithm, and the time complexity will be $O(\tau(|V|+|Q|)\log|Q|+|E|)$.

Example 9: Consider the two graphs in Figure 5, $\tau = 2$, and $q = 2$. Global label filtering yields a lower bound of 2; count filtering requires the two graphs share at least 2 q-grams, while they do share C-C-C *and* C-C-C*; Minimum edit filtering only gives a lower bound of 2 for both r and s. Therefore the pair can pass these three filters. The two bounded regions in the figure show the two components formed by jointing the mismatching q-grams in r. The edit operations on the left component will be 1 (from minimum edit filtering), and 2 on the right component (from local label filtering). Therefore the pair can be pruned.*

VI. Verification Algorithm

Our verification algorithm consists of two parts: (1) multiple filters that quickly prune unpromising candidates; and (2) computation of graph edit distance. We introduce both parts in detail.

A. Integrating Multiple Filters

The verification algorithm for GSimJoin is shown in Algorithm 6. The candidates are verified through three filters in succession: global label filtering (Lines 3 – 4), count filtering (Lines 5 – 6), and local label filtering (Lines 7 – 9). Those still survive will be verified through the expensive graph edit distance computation. The CompareQGrams algorithm in Line 5 extracts the mismatching q-grams from both r to s and s to r, returned in Q'_r and Q'_s respectively. In addition, we carefully compute the numbers of mismatching q-grams in both directions without double-counting, and return them in ϵ_2 and ϵ_3.

Algorithm 6: Verify(r, A)

```
Input   : $r$ is a graph; $A$ is map indicating $r$'s candidates.
Output  : $S = \{ \langle r, s\rangle \mid ged(r, s) \le \tau \}$.
1  $S \leftarrow \emptyset$;
2  for each $s$ such that $A[s] =$ true do
3      $\epsilon_1 \leftarrow \Gamma(L_V(r), L_V(s)) + \Gamma(L_E(r), L_E(s))$
       ;                       /* global label filtering */
4      if $\epsilon_1 \le \tau$ then
5          $(Q'_r, Q'_s, \epsilon_2, \epsilon_3) \leftarrow$ CompareQGrams$(Q_r, Q_s)$
           ;                           /* count filtering */
6          if $\epsilon_2 \le \tau \cdot D_{path}(r)$ and $\epsilon_3 \le \tau \cdot D_{path}(s)$ then
7              $\epsilon_4 \leftarrow$ LocalLabelFilter$(Q'_r, s)$
               ;               /* local label filtering */
8              $\epsilon_5 \leftarrow$ LocalLabelFilter$(Q'_s, r)$
               ;               /* local label filtering */
9              if $\epsilon_4 \le \tau$ and $\epsilon_5 \le \tau$ then
10                 edit $\leftarrow$ GraphEditDistance$(r, s)$;
11                 if edit $\le \tau$ then
12                     $S \leftarrow S \cup \{ \langle r, s\rangle \}$;
13 return $S$
```

B. Graph Edit Distance Computation

Most widely used exact approaches for computing graph edit distance are based on A* algorithm [11]. In this section, we briefly review a state-of-the-art approach for computing exact ged [20], and then see how our mismatch filtering techniques can be employed to speed up the algorithm.

A* explores the space of all possible vertex mappings between two graphs in a best-first search fashion with a function (denoted $f(x)$) established to determine the order in which the search visits vertex mappings. $f(x)$ is a sum of two functions: (1) the distance from the initial state to the current state (denoted $g(x)$); and (2) a heuristic estimate of the distance from the current state to the goal (denoted $h(x)$). A* maintains states in a priority queue, and guarantees the path to the goal is shortest when the goal is popped from the queue, if the $h(x)$ function is *admissible*; i.e., $h(x)$ is lower than or equal to the real distance from the current to the goal.

With no vertex mapped in the initial state, we form a new state in each step by mapping a vertex in r to either a vertex in s, or none to imply a vertex deletion. The goal is to map all the vertices in r. $g(x)$ is the graph edit distance between the two partial graphs corresponding to current vertex mapping. For $h(x)$, [20] gives a lower bound of the graph edit distance between the remaining parts with bipartite matching. The original algorithm is designed for weighted graph edit distance. For our unweighted version, $h(x)$ becomes exactly the result of global label filtering.

$$g(x) = ged(r_p, s_p);$$
$$h(x) = \Gamma(L_V(r_q), L_V(s_q)) + \Gamma(L_E(r_q), L_E(s_q)).$$

r_p consists of the vertices that have been mapped and the edges connecting them, while r_q consists of the vertices unmapped yet as well as their resident edges.

1) Exploiting Minimum Edit Filtering: Although A* algorithm adopts a best-first search scheme to efficiently compute graph edit distance, it does not discuss the impact of search order on the efficiency of the algorithm. Due to the removal of unpromising candidates with multiple filters, the pairs verified by A* algorithm are very likely to resemble though they may not satisfy the graph edit distance constraint. The isomorphic part of the graphs do not incur any edit operations, and therefore the goal cannot be found until very late stage of the process if we start searching from this part. In contrast, the process ends more quickly if we start with the part that needs edit operations.

Recall the mismatching q-grams identified by CompareQGrams algorithm. The mismatching q-grams indeed contribute edit operations and hence should be favored. Algorithms 7 exploits this idea and determines the order of vertices to be mapped by the A* algorithm. The vertices contained by at least one mismatching q-gram are put before the others. In the interest of connectivity, we break tie by mapping vertices in the order of spanning tree, so as to expedite the discovery of edge edit operations. Using such order leverages the connectivity of a graph and can quickly find edge edit operations. E.g., assume in r, u and v are adjacent vertices in the spanning tree, and they are mapped to u' and v' in s, respectively. An edge edit operation will occur if there is no edge between u' and v' in s.

2) Exploiting Local Label Filtering: Any lower bound of graph edit distance can serve as the heuristic estimate $h(x)$

Algorithm 7: DetermineVertexOrder(r, Q'_r)

```
Input  : $r$ is graph;
         $Q'_r$ is a multiset of mismatching $q$-grams from $r$ to $s$.
Output : An array of vertices
         that the A* algorithm will find mapping in order.
1 $M \leftarrow []$;
2 $C \leftarrow$ the connected components formed by $Q'_r$;
3 for each $c_i \in C$ do
4     Insert vertices in $c_i$ into $M$ in the order of spanning tree;
5 Insert the vertices not contained by
  any mismatching $q$-gram into $M$ in the order of spanning tree;
6 return $M$
```

to render the A* algorithm admissible. We consider not only global label filtering but also local label filtering in $h(x)$. The mismatching q-grams in the remaining graphs composed of unmapped vertices are first extracted, and then sent into local label filtering to get lower bounds of graph edit distance between the two remaining graphs. Algorithm 8 provides the pseudocode of the algorithm. Note that we compute mismatching q-grams from both r_q to s_q and s_q to r_q, and hence have two lower bounds from local label filtering. The lower bound from global label filtering is also considered, and the maximum of the three is returned as the result of heuristic estimate.

Algorithm 8: EstimateDistance(r_q, s_q)

```
Input  : $r_q$ and $s_q$ are two graphs consist of unmapped vertices
Output : A lower bound of $ged(r_q, s_q)$
1 $\epsilon_1 \leftarrow \Gamma(L_V(r_q), L_V(s_q)) + \Gamma(L_E(r_q), L_E(s_q))$;
2 $(Q'_r, Q'_s) \leftarrow$ CompareQGrams$(r_q, s_q)$;
3 $\epsilon_2 \leftarrow$ LocalLabelFilter$(Q'_r, s_q)$;
4 $\epsilon_3 \leftarrow$ LocalLabelFilter$(Q'_s, r_q)$;
5 $h \leftarrow \max(\epsilon_1, \epsilon_2, \epsilon_3)$;
6 return $h$
```

VII. Experiments

In this section, we report experiment results and our analyses.

A. Experiment Setup

The following algorithms are used in the experiment.

- **GSimJoin** is our proposed algorithm that utilizes path-based q-grams.
- **κ-AT** is a state-of-the-art algorithm based on tree-based q-grams [28]. We implemented this algorithm and applied size filtering, prefix filtering, and global label filtering successively to find the candidates that need graph edit distance verification, as they also work for tree-based q-grams. We ran κ-AT algorithm with different q-gram lengths and found $q = 1$ yields the smallest candidate size as well as the best runtime performance under all our threshold settings, and consequently we choose $q = 1$ for κ-AT algorithm.
- **AppFull** is another state-of-the-art algorithm based on star structure [36]. In order to make it support graph similarity joins, we run the the binary code in a nested loop join mode. It iterates through the dataset and selects each graph as a query, and the corresponding database contains all the graphs with smaller identifiers. The filtering time for each query is then summed up as the total filtering time. Edge labels in datasets are omitted when comparing with AppFull as the binary code ignores edge labels.

All experiments were carried out on a Quad-Core Intel Xeon Processor X3330@2.66GHz with 4GB RAM. The operating system is Debian 5.0.6. All algorithms with source codes were coded in C++. We compiled them using GCC 4.3.2 with `-O3` flag, and all the algorithms were run in main memory.

With respect to q-gram storage, we assume the label of a vertex or an edge takes 1 byte. Since a q-gram is a path of length q, $2q + 1$ bytes are needed to store a q-gram if we concatenate the labels of the vertices and edges in the path. In our implementation, we choose to hash a q-gram into a 4-byte integer. This not only controls the index size, but also speeds up equality checking. The only downside is the existence of false positives within candidates due to hash collision. This will not affect correctness but only efficiency.

We selected two publicly available real datasets with different data distributions.

- **AIDS** is the antivirus screen compound dataset from the Developmental Theroapeutics Program in NCI/NIH [4]. It contains 42,687 chemical compounds. We randomly sample 4,000 graphs to make up the dataset used in the experiment.
- **PROTEIN** is the protein database from the Protein Data Bank [5] and labeled with their corresponding enzyme class labels. It contains 600 protein structures. Vertices represent secondary structure elements and are labeled with their types (helix, sheet, or loop). Edges are labeled to indicate whether the two elements are neighbors along the amino acid sequence or neighbors in space within the protein structure.

Statistics about the datasets are listed in Table I. AIDS is composed of sparse graphs while those in PROTEIN are denser.

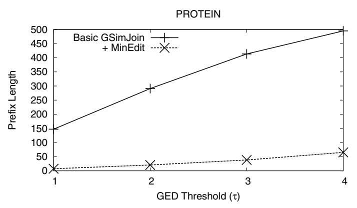
TABLE I
Statistics of the Datasets

Dataset	$\|R\|$	avg $\|V\|$	avg $\|E\|$	avg $\|l_V\|$	avg $\|l_E\|$
AIDS	4,000	25.6	27.5	44	3
PROTEIN	600	32.6	62.1	3	2

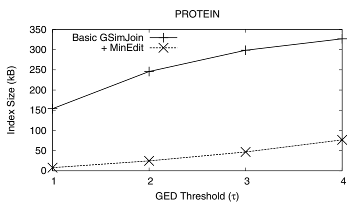
We measure (1) the average length of prefixes for GSimJoin and κ-AT; (2) the index size for GSimJoin and κ-AT; (3) the candidates formed after probing inverted index for GSimJoin and κ-AT (denoted **Cand-1**); (4) the candidates that need graph edit computation for GSimJoin and κ-AT, and the pairs of graphs that pass both lower bound and upper bound tests of AppFull (denoted **Cand-2**); and (5) the running time.

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

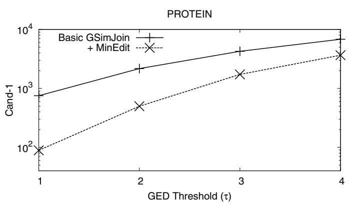
[5]http://www.iam.unibe.ch/fki/databases/iam-graph-database/download-the-iam-graph-database



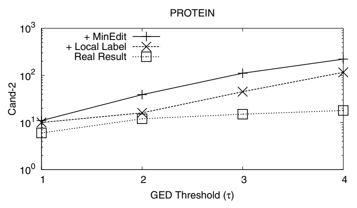
(a) PROTEIN, Prefix Length



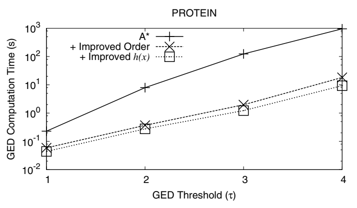
(b) PROTEIN, Index Size



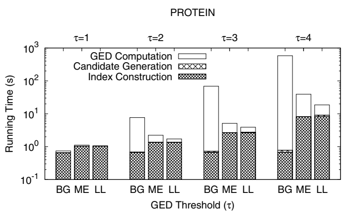
(c) PROTEIN, Cand-1



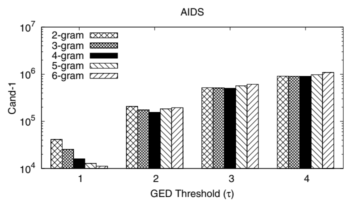
(d) PROTEIN, Cand-2



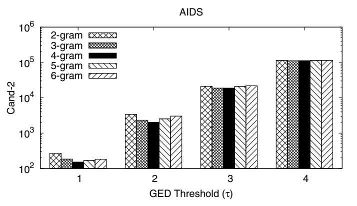
(e) PROTEIN, GED Computation Time



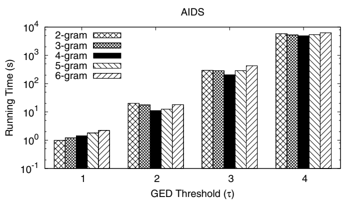
(f) PROTEIN, Total Running Time



(g) AIDS, Cand-1



(h) AIDS, Cand-2



(i) AIDS, Total Running Time

Fig. 6. Experiment Results - I

B. Evaluating Filters

In order to evaluate the effectiveness of our filtering techniques, we use the term "Basic GSimJoin " for the GSimJoin algorithm without minimum edit or local label filtering. "+ MinEdit" denotes applying minimum edit filtering to compute the prefix length, and "+ Local Label" denotes further applying local label filtering; i.e., the complete GSimJoin algorithm.

We first study the effect of minimum edit filtering. Figure 6(a) shows the average prefix lengths of Basic GSimJoin and + MinEdit on PROTEIN dataset with $q = 3$ and varying edit distance threshold. + Local Label has the same prefix length of + MinEdit, so we do not show it in this figure. The prefix lengths from both algorithms grow steadily when the threshold increases. The prefix length has been substantially reduced after applying minimum edit filtering, and the reduction is more significant when τ is small. When $\tau = 1$, the prefix length can be reduced by 95%. As index size is influenced by prefix length, we plot the memory consumed for indexing by the two algorithms in Figure 6(b). Both algorithms need small amount of memory and exhibit a similar trend as on prefix length. The memory consumed by + MinEdit is only 76.6k when τ is as large as 4. The number of Cand-1's is also influenced by prefix length, as plotted in Figure 6(c) in logarithmic scale. The Cand-1 size can be reduced by as much as 88% when τ is 1.

As for local label filtering, figure 6(d) compares the number of Cand-2s produced by + MinEdit and + Local Label on PROTEIN. The number of real join results is also shown. Local label filtering results in remarkable reduction on Cand-2s, which can be up to 62%.

C. Evaluating Graph Edit Distance Computation

To evaluate the optimization in graph edit distance computation, we choose the candidate pairs generated by + Local Label with the parameters $q = 4, \tau = 4$, and verify them with different algorithms. The A* algorithm proposed in [20] is labeled as "A*". Minimum edit filtering is exploited to improve the search order, and the result algorithm is labeled "+ Improved Order". Local label filtering is further applied to improve heuristic estimate $h(x)$ and labeled "+ Improved $h(x)$". Figure 6(e) reports the graph edit distance computation time for the three algorithms with varying τ. We observe that the optimizations can enhance the time efficiency of the ged computation, and the margin is more significant with larger τ's.

Combining the filtering algorithms and ged computation algorithms according to the techniques employed, we show the total running time and decompose it into different phases in Figure 6(f). The notations in the figure denote the following combinations:

- BG: Basic GSimJoin / A*;
- ME: + MinEdit / + Improved Order;
- LL: + Local Label / + Improved $h(x)$.

BG has smaller index construction and candidate generation time, but becomes less competitive for large τ's in terms of total running time due to its large Cand-2 size and less efficient ged computation. LL can be up to 2.1 times faster than ME and 31.4 times faster than BG.

D. Evaluating q-gram Length

We ran GSimJoin algorithm on AIDS dataset with q-gram length varying in the range [2, 6], and plot the Cand-1, Cand-2, and running time in Figures 6(g) –6(i).

With respect to Cand-1 and Cand-2, the general trend is that the candidate size first drops with an increasing q-gram length, reaches the bottom at a q of 3 or 4, and then rebounds. There are several factors contributing to this trend: (1) Small q indicates a small q-gram domain, and hence the inverted list of a q-gram can be fairly long. This will lead to a large candidate size, especially when q is 2. (2) Large q indicates a long prefix length. We have to probe more inverted lists and hence it will increase the candidate size. The second factor explains why the candidate size rebounds for long q-grams. It can be seen when q is 6, the candidate size is the actually the largest for most threshold settings.

The trend of candidate size reflects the running time under varying q. As can be expected from candidate size, $q = 3$ or 4 will be the most competitive in total running time. The figure shows the best runtime performance is achieved when q is 4 for $\tau \in [2, 4]$. The only exception is, when $\tau = 1$, $q = 2$ is the most time-efficient setting. This is because the generation of q-grams and index construction take most of running time for this threshold setting.

In the rest of the experiment, we use $q = 4$ on AIDS and $q = 3$ on PROTEIN as they are the most time-efficient.

E. Comparing with κ-AT

We compare GSimJoin with κ-AT algorithm on both datasets.

The average prefix lengths of both algorithms are shown in Figures 7(a) and 7(b). In spite of a longer prefix on AIDS, GSimJoin has more average number of q-grams in a graph. For example, κ-AT's prefix length is 8.2 when τ is 1 and the average number of q-grams in a graph is 25.6, while GSimJoin's prefix length is 8.9 and the average number of q-grams is 71.5. This means κ-AT requires two graphs have an average of $25.6 - 8.2 + 1 = 18.4$ common q-grams to become a candidate, while GSimJoin needs an average of $71.5 - 8.9 + 1 = 63.6$ common q-grams. Note that the q-gram length is 1 for κ-AT; i.e., the count filtering of κ-AT is the *tightest* among all its q settings. On PROTEIN, the prefix length of GSimJoin is even shorter than κ-AT under some parameter settings. Both algorithms are competitive in index sizes, as shown in Figures 7(c) and 7(d).

Figures 7(e) – 7(h) give the Cand-1 and Cand-2 sizes of the two algorithms. GSimJoin performs better than κ-AT on both Cand-1 and Cand-2 sizes. There are three main factors: (1) 4-grams based on paths are more selective than 1-grams based on trees. This results in a less number of Cand-1s for GSimJoin. (2) GSimJoin's count filtering constraint is stricter than κ-AT's. (3) GSimJoin employs local label filtering to further prune candidates. The last two factors contribute to GSimJoin's advantage on Cand-2 numbers. The running time of both algorithms are shown in Figures 7(i) and 7(j) ("AT" and "GS" are short for κ-AT and GSimJoin respectively). κ-AT shows better index construction time and candidate generation time as GSimJoin needs minimum edit and local label filtering to build index and prune candidates. However, GSimJoin is always better than κ-AT in terms of total running time, and the gap is more substantial under large τ settings. The speed-up on AIDS is 6.6x and 80.6x on PROTEIN. The latter one showcases the superior time advantage of GSimJoin on denser graphs.

F. Comparing with AppFull

We compare GSimJoin with AppFull on both datasets and plot the number of Cand-2s and running time in Figures 7(k) – 7(n) ("AF" and "GS" are short for AppFull and GSimJoin respectively). Since the released binary code from the authors of [36] reports only the number of candidates and and filtering time, we cannot conduct the graph edit distance computation for AppFull. Its filtering time will be used as a *lower bound* of total running time and compared with GSimJoin's total running time.

AppFull's Cand-2 size is smaller than GSimJoin. The main reason is that its bipartite matching can get tight lower/upper bounds of graph edit distance. AppFull exhibits almost constant filtering time under different τ settings due to the all-pair bipartite matching and lack of index. Although the running time of GSimJoin grows when we move towards larger τ's, it is still always faster than AppFull on AIDS. For threshold settings in [1, 3] on PROTEIN, GSimJoin is also more time-efficient. AppFull spends less time on PROTEIN when τ is 4, however, GSimJoin can output all the answers while AppFull generates a set of candidates that still need verification.

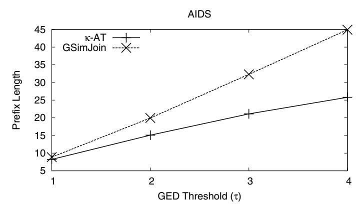
G. Varying Dataset Sizes

We compare the scalability of GSimJoin and κ-AT algorithms on AIDS dataset with a fixed τ of 2. We randomly sampled about 20% to 100% from the 4,000 graphs so that the data and result distribution remain approximately the same with the whole dataset. We show the square root of the running time in Figure 7(o).

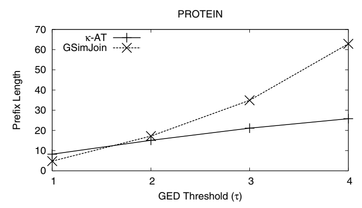
It is not surprising to notice that the running time of both algorithms grow quadratically, given the fact that the real join result size has a quadratic growth. The numbers of real join results are 5, 24, 41, 79, and 129 for the five scales, respectively. Our GSimJoin also demonstrates advantage over the κ-AT as its growth rate is slower.

VIII. Related Work

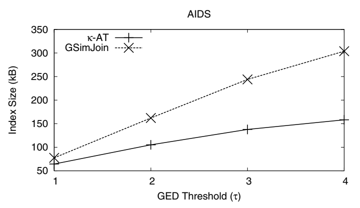
Similarity join has been extensively studied due to its importance in many applications domains, including record linkage, data cleaning, multimedia applications, and phenomena detection on sensor networks. As a consequence, similarity joins on various data types become the research theme of many recent literature on text data [10], probabilistic data [17], stream data [16] and so forth.



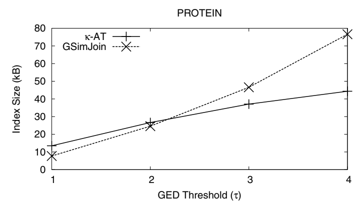
(a) AIDS, Prefix Length



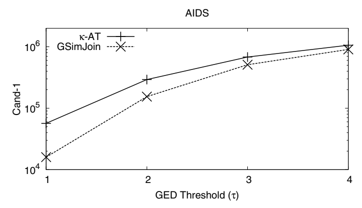
(b) PROTEIN, Prefix Length



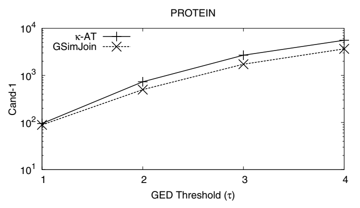
(c) AIDS, Index Size



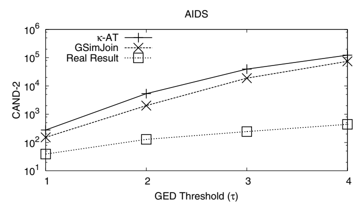
(d) PROTEIN, Index Size



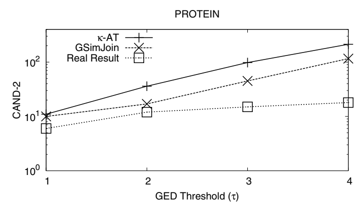
(e) AIDS, Cand-1



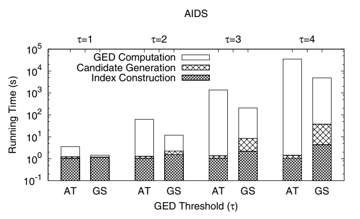
(f) PROTEIN, Cand-1



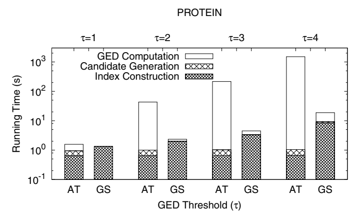
(g) AIDS, Cand-2



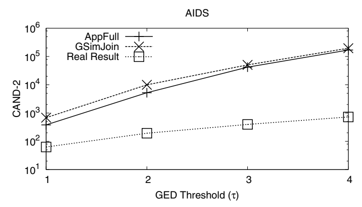
(h) PROTEIN, Cand-2



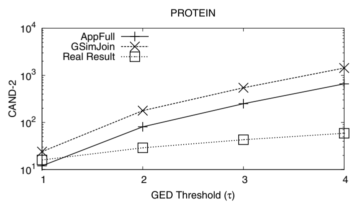
(i) AIDS, Total Running Time



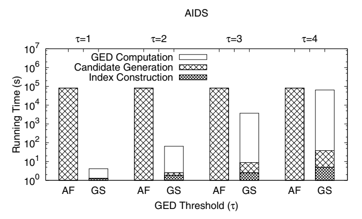
(j) PROTEIN, Total Running Time



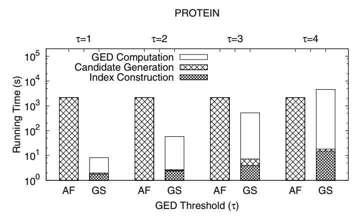
(k) AIDS, Cand-2



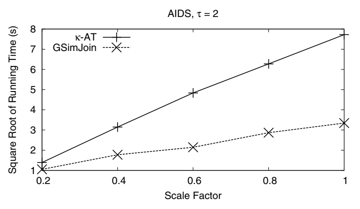
(l) PROTEIN, Cand-2



(m) AIDS, Running Time



(n) PROTEIN, Running Time



(o) AIDS, Total Running Time

Fig. 7. Experiment Results - II

Similarity join on strings with edit distance constraints is well-studied by database communities. q-gram technique is widely used for string similarity matching, especially for edit distance constraints [10], [31]. Apart from fixed-length q-grams, variable length grams (VGRAMs) are also adopted [15], [35].

To our best knowledge, there is no research literature directly targeting similarity join on graph data. Nonetheless, a closely related topic is structure similarity selection over graphs, which keeps receiving considerable attention recently. Closure-Tree is put forward to identify k data graphs that are most nearly isomorphic to the query graph [12]. To formalize a general definition of structure similarity, graph edit distance is employed as a metric of the difference between graphs [36]. Latest advance in graph similarity selection is from the idea of using κ-AT [28] to prune the false positive graphs, where the similarity definition based on edit distance is followed. Inspired by q-gram technique, it builds index by decomposing each data graph into κ-ATs, and filters data graphs by comparing the threshold with the lower bound of the edit distance derived from the index.

Subgraph similarity search is to retrieve graphs that approximately contain a query graph. To facilitate such queries, a DAG-structured index incorporating a hash table is introduced to solve the problem with strong constraints [30]. Grafil [34] develops a feature based pruning technique to conduct subgraph similarity search, and the similarity is defined as the number of missing edges with respect to the maximum common subgraph. Concerning the particular interest on connected subgraphs, GrafD-index [23] exploits a set of effective pruning and validation rules to tackle the problem of searching for connected structures that are similar to the maximum connected common subgraph. As the counterpart, supergraph similarity search is also investigated; i.e. to retrieve graphs that are approximately contained by a given query graph [24].

Another line of related research focuses on graph edit distance computation. So far the fastest exact solution is credited to an A*-based algorithm incorporating a bipartite heuristic [20]. To render the matching process less computationally demanding, a number of approximate methods are proposed to find suboptimal answer [26], [3], [9], [19]. Another type of solution is to convert it to binary linear programming and compute the bounds of the answer [13].

IX. Conclusion

In this paper, we study the problem of graph similarity join with edit distance constraints. Unlike previous methods which use trees or star structures to find candidates, we propose a method exploiting the number of common fixed-length paths between pairs of graphs. An algorithm, GSimJoin, is designed to find answers to graph similarity join. Two additional filtering techniques are developed to deal with both scattered and clustered edit operations as well as facilitate the graph edit distance computation. Finally, comprehensive experiments performed on real datasets demonstrate that the new algorithm outperforms alternatives.

Acknowledgement. This work was partially done at East China Normal University when the corresponding author was taking a Chinese academic program and the first author was visiting there, and supported by NSFC61021004. The work was partially supported by ARC DP120104168, DP110102937, and DP0987557. The fourth author was also supported in partial by ARC DP0987273 and DP0881779.

References

[1] R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging, and its application to diatom idenfication. In *GbRPR*, pages 95–106, 2003.
[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1st edition edition, May 1999.
[3] M. C. Boeres, C. C. Ribeiro, and I. Bloch. A randomized heuristic for scene recognition by graph matching. In *WEA*, pages 100–113, 2004.
[4] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245 – 253, 1983.
[5] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.
[6] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han. Mining graph patterns efficiently via randomized summaries. *PVLDB*, 2(1):742–753, 2009.
[7] C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang, and X. Gu. Towards graph containment search and indexing. In *VLDB*, pages 926–937, 2007.
[8] J. J. Cottell, J. O. Link, S. D. Schroeder, J. Taylor, W. C. Tse, R. W. Vivian, and Z.-Y. Yang. Antiviral compounds, patent WO2009005677, January 2009.
[9] S. Fankhauser, K. Riesen, and H. Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *GbRPR*, pages 102–111, 2011.
[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.
[11] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, Feb. 1968.
[12] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, page 38, 2006.
[13] D. Justice and A. O. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.
[14] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.
[15] C. Li, B. Wang, and X. Yang. VGRAM: Improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, 2007.
[16] X. Lian and L. Chen. Efficient similarity join over multiple stream time series. *IEEE Trans. Knowl. Data Eng.*, 21(11):1544–1558, 2009.
[17] X. Lian and L. Chen. Set similarity join on probabilistic data. *PVLDB*, 3(1):650–659, 2010.
[18] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *SIGMOD*, 2011.
[19] R. Raveaux, J.-C. Burie, and J.-M. Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5):394–406, 2010.
[20] K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *MLG*, 2007.
[21] A. Robles-Kelly and E. R. Hancock. Graph edit distance from spectral seriation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):365–378, 2005.
[22] A. Sanfeliu and K.-S. Fu. A Distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, 13(3):353–362, 1983.
[23] H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang. Connected substructure similarity search. In *SIGMOD Conference*, pages 903–914, 2010.
[24] H. Shang, K. Zhu, X. Lin, Y. Zhang, and R. Ichise. Similarity search on supergraph containment. In *ICDE*, pages 637–648, 2010.
[25] P. Slavík. A tight analysis of the greedy algorithm for set cover. In *STOC*, pages 435–441, 1996.
[26] S. Sorlin and C. Solnon. Reactive tabu search for measuring graph similarity. In *GbRPR*, pages 172–182, 2005.
[27] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *ICDE*, pages 963–972, 2008.
[28] G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1, 2010.
[29] J. Wang, J. Feng, and G. Li. Trie-join: Efficient trie-based string similarity joins with edit. In *VLDB*, 2010.
[30] D. W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *ICDE*, pages 976–985, 2007.
[31] C. Xiao, W. Wang, and X. Lin. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB*, 1(1):933–944, 2008.
[32] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.
[33] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD Conference*, pages 335–346, 2004.
[34] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD Conference*, pages 766–777, 2005.
[35] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *SIGMOD Conference*, pages 353–364, 2008.
[36] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.
[37] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. B^{ed}-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD Conference*, pages 915–926, 2010.
[38] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, pages 938–949, 2007.