THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS IEICE Technical Report

A Study on Graph Similarity Search

Haichuan Shang Masaru Kitsuregawa
Institute of Industrial Science
University of Tokyo
{shang,kitsure}@tkl.iis.u-tokyo.ac.jp

ABSTRACT

Graph similarity search is to retrieve graphs that approximately contain a given query graph. It has many applications, e.g., detecting similar functions among chemical compounds. The problem is challenging as even testing subgraph containment between two graphs is NP-complete. Hence, existing techniques adopt the filtering-and-verification framework with the focus on developing effective and efficient techniques to remove non-promising graphs.

Nevertheless, existing filtering techniques may be still unable to effectively remove many "low" quality candidates. To resolve this, in this paper we propose a novel indexing technique to index graphs according to their "distances" to features. We then develop lower and upper bounding techniques that exploit the index to (1) prune non-promising graphs and (2) include graphs whose similarities are guaranteed to exceed the given similarity threshold. Considering that the verification phase is not well studied and plays the dominant role in the whole process, we devise efficient algorithms to verify candidates. A comprehensive experiment using real datasets demonstrates that our proposed methods significantly outperform existing methods.

1. INTRODUCTION

Graphs have a wide range of applications including bioinformatics, chemistry, social networks, pattern recognition, software engineering. In these applications, graphs are used to model complex structured data and relationships. For example, graphs have been used to model and store chemical compounds. UML and ER diagrams are other examples. There has been a considerable effort, from both database and data mining communities, in developing techniques for managing, processing, and analyzing graph databases, including graph pattern discovery structure-based graph queries etc.

The *substructure search* problem, also called *subgraph containment query*, is that for a graph database and a given query graph, we want to find all data graphs which contain the query graph. Figure 2 shows a sample graph database. Suppose that q_1 in Figure 1 is used as a query graph; then $\{g_3\}$ is the result of the subgraph search. Such queries are very useful for an exploration purpose in many applications (e.g., drug design, computer vision and pattern recognition, and medical images) to extract and identify a small set of molecules and graph models for further analysis. A common

Figure 1: Query Graphs

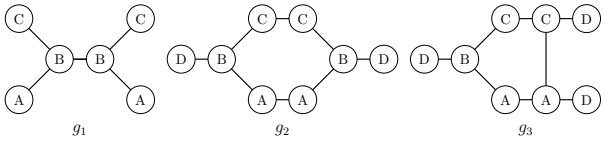
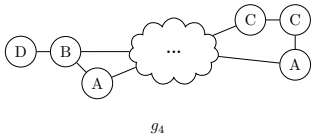


Figure 2: A Sample Graph Database

problem is that in many occasions, there could be no match for such an exploratory query; for instance, q_2 in Figure 1 is not contained by any graph in Figure 2. In stead of refining a query graph manually by users, [7] proposes to ask systems to find out graphs that "nearly" contain the query graph; it is formulated as the *substructure similarity search*, also called *subgraph similarity search*. To capture global structure information, the subgraph similarity search problem is defined [7] as the problem of detecting the Maximum Common Subgraph (MCS) between the query graph and the database graphs, and the measure of similarity is then based on the difference of the query graph and the MCS. It is well known that detecting MCS is NP-complete [2]. Hence, existing techniques [7, 8], to support the subgraph similarity search, follow the *filtering-and-verification* paradigm with the focus on removing non-promising graphs as many as possible in filtering to avoid expensive verification.

Connected Subgraph Similarity Search. MCS may include many low-quality results in subgraph similarity search. Intuitively, it is possible that different parts of a query are mapped to very different locations in a data graph g which are far away from each other. For example, if q_1 in Figure 1 is used and we are allowed to miss at most 2 edges, then the MCS-based similarity search will return g_4 in Figure 3 as a result. Clearly, such a result is usually not desirable from users. This phenomenon is not uncommon in subgraph similarity search, as data graphs are usually much larger than a query graph in typical settings. Motivated by this, in this paper we investigate the problem of substructure similarity search based on maximum connected common subgraphs (MCCS).

The filtering techniques [7, 8] inherently do not provide



g_4

Figure 3: Cloud Contains a Large Number of Nodes

a very effective support to connected subgraph similarity search; for instance, it is impossible to exclude the data graph g_4 in Figure 3 from candidate graphs by these two existing filtering techniques. Moreover, the verification phase is not studied in the existing work [7, 8] though it plays the dominant role in the whole computation. In fact, to the best of our knowledge there is no existing algorithm to conduct verification for the MCCS-based subgraph similarity search.

Contributions. Motivated by these, we develop a novel index technique, GrafD-index, which indexes data graphs according to their distances (to be defined in Section 2) to a feature (for each feature). We then characterize a tight condition under which triangular inequality holds for defined distance functions. Consequently, a novel lower-bounding technique is developed to prune data graphs that are guaranteed not in the query result. We also develop an upper-bounding technique to perform early validation to include data graphs into the query result without any costly verification. Both pruning and validation are supported efficiently by the GrafD-index. Finally, we develop an efficient verification algorithm that is "optimized" to share the computation. Our contributions can be summarized as follows:

1. We develop a novel index technique, GrafD-index, to effectively index data graphs according to their MCCS-based distances to features.

2. We formally prove triangular inequality holds on the MCCS-based distance function under a tight sufficient condition based on graph connectivity.

3. Based on the GrafD-index and the triangular inequality, new *pruning* and *validation* techniques are developed to quickly identify non-answers and sure-answers.

4. We develop novel, efficient algorithms to verify whether a candidate graph satisfies the similarity threshold against the query graph.

Comprehensive experiments using real datasets demonstrate that our techniques are efficient and scalable, and significantly outperform the (only) two existing filtering techniques [7, 8]. They also indicate that our total computation (filtering, validation, and verification) is more efficient than the filtering technique in [8] for high-similarity search. Our filtering and validation techniques significantly reduce (up to 80% size reduction) the size of the candidate set by Grafil [7]. To further evaluate the effectiveness of our filtering techniques, our experiment results show that the total costs of our techniques are always significantly lower than those of Grafil combining with our verification techniques.

The rest of the paper is organized as follows. Section 2 presents problem definitions and the preliminaries. Section 3 introduces pruning and validation rules, as well as the framework of our approach. Section 4 reports the experimental results. The conclusion is given in Section 5.

2. BACKGROUND INFORMATION

The research in this paper is focused on undirected *vertex-labeled* connected graphs.[1] Given a set of labels, Σ_V, a graph is denoted by $G = (V, E, l)$ where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, and l is a labeling function: $V \rightarrow \Sigma_V$. We denote the vertex set and the edge set of a graph g by $V(g)$ and $E(g)$, respectively. $l(u)$ denotes the label of u. $|V(g)|$ and $|E(g)|$ represent the number of vertices and edges, respectively. For presentation simplicity, an undirected vertex-labeled graph is hereafter abbreviated to a graph.

2.1 Problem Statement

Substructure Similarity Search. Subgraph isomorphism and maximum connected common subgraphs (MCCS) are defined as follows.

Definition 1. (Subgraph Isomorphism) Given two graphs $g' = (V', E', l')$ and $g = (V, E, l)$, g' is *subgraph-isomorphic* to g, denoted as $g' \subseteq_{\mathcal{F}} g$, if there is an injective function $\mathcal{F}: g' \rightarrow g$ such that

1. $\forall v \in V'$, $\mathcal{F}(v) \in V(g)$ such that $l'(v) = l(\mathcal{F}(v))$.
2. $\forall (u, v) \in E'$, $(\mathcal{F}(u), \mathcal{F}(v)) \in E$.

$g' \subseteq_{\mathcal{F}} g$ is used to denote that a graph g' is subgraph-isomorphic to g under the function $\mathcal{F}$ where g' is called a *subgraph* of g and g is also called a *supergraph* of g'; we may also simply say that g contains g'. $g' \subseteq_{\mathcal{F}} g$ is abbreviated to $g' \subseteq g$ if there is no ambiguity. Note that more than one subgraph isomorphic mapping may exist between g' and g.

Definition 2. (Maximum Common Connected Subgraph - MCCS) Given two graphs g_1 and g_2, the maximum common connected subgraph of g_1 and g_2 is the largest *connected* subgraph of g_1 that is subgraph-isomorphic to g_2, denoted as $mccs(g_1, g_2)$.

Note that in Definition 2, the size of a graph is measured by the number of edges.

Definition 3. (Query Relaxation Distance) Given a query graph q and a data graph g, the query relaxation distance based on MCCS is defined as,

$$dist(q, g) = |E(q)| - |E(mccs(q, g))|.$$

Definition 4. (Subgraph Similarity Search) Given a graph database $D = \{g_1, g_2, \ldots, g_n\}$, a query graph q, and a threshold σ, the subgraph similarity search problem is to retrieve all the graphs $g_i \in D$ with $dist(q, g_i) \leq \sigma$. σ is also called a distance threshold.

Note that the distance is asymmetric as $dist(q, g) \neq dist(g, q)$ unless $|q| = |g|$. [7] defines the query relaxation distance based on MCS and the *relaxation ratio* $\frac{dist(q,p)}{|q|}$ is used for subgraph similarity search. Clearly, techniques for computing relaxation distances can be immediately applied to computing relaxation ratios.

Problem Statement. In this paper, we will develop efficient algorithms to conduct subgraph similarity search based on the MCCS-based query relaxation distance.

[1]The developed techniques can be immediately extended to edge-labeled and/or directed graphs.

2.2 Preliminaries

Grafil. Grafil [7] is developed to support efficient subgraph similarity searches and follows the *filtering-verification* query processing paradigm. It provides a feature-based index [3, 6] to effectively filter non-promising data graphs. Features could be paths [3], trees [9], or subgraphs [6].

As shown in Figure 4(a), a *feature-graph matrix* M is constructed by Grafil, which stores the number of the subgraph isomorphic mappings from a feature to a data graph: $M_{ij} = |\{\mathcal{F}|\ f_i \subseteq_{\mathcal{F}} g_j\}|$, where f_i is the i-th feature, g_j is the j-th data graph and $\mathcal{F}$ is a subgraph-isomorphic mapping. When a query graph q is issued, a *binary edge-feature-mapping matrix* is built on-the-fly by computing all the subgraph isomorphic mappings from each feature to the query graph.

As shown in Figure 4(b), the number of columns is the total number of feature mappings found in q, and each cell in the edge-feature-mapping matrix indicates whether the edge is involved in a particular mapping. For instance, the first column shows that feature f_1 can be mapped to edges $\{e_1, e_2\}$ of q; feature f_2 has two mappings $f_{2(1)}$ and $f_{2(2)}$ to q.

Grafil calculates the maximum number (an upper-bound), denoted by d_{max}, of feature mappings that can be *missed* by removing σ edges in q. Then, for each data graph g, Grafil calculates the number of feature mappings to q but not to g, denoted by $d(q, g)$ and called *outstanding number*. If $d(q, g) \leq d_{max}$, then g is included as a candidate graph.

	g_1	g_2	g_3
f_1	2	0	2
f_2	0	3	0
f_3	2	0	1
f_4	0	0	1

(a) Feature Graph matrix

	f_1	$f_{2(1)}$	$f_{2(2)}$	f_4
e_1	1	1	1	0
e_2	1	1	0	1
e_3	0	0	1	0

(b) Edge Feature Matrix

Figure 4: Matrices Used in Grafil

EXAMPLE 1. *Consider the two matrices in Figures 4(a) and 4(b), respectively. Let $\sigma = 1$. It can be verified that at most 3 feature mappings may be missed by removing one edge; thus $d_{max} = 3$. Note that the query graph contains f_1 once, f_2 twice, and f_4 once. Regarding g_1, g_1 contains f_1 twice and f_3 twice. Thus, the outstanding number is 0 regarding f_1, 2 regarding f_2, 0 regarding f_3, and 1 regarding f_4, respectively. Summing them together gives 3. Since $3 \leq d_{max}$, g_1 is a candidate graph. Similarly, g_2 and g_3 are also kept as the candidate graphs.*

QuickSI. An efficient verification algorithm, QuickSI [5], is developed to determine whether there is a subgraph isomorphic mapping from q to g.

Clearly, a mapping $\mathcal{F}$ of q to g is fixed if the mapping $\mathcal{F}$ from all vertices of q to g is determined. Nevertheless, a vertex in q may be mapped to many vertices in g with the same label. Consequently, there may be too many feasible combinations to consider; for instance, if each vertex from q has the same label with that of m vertices in g, then we need to consider n^m combinations in the worst case. Instead of trivially enumerating mappings from $V(q)$ to $V(g)$, QuickSI enumerates mappings from a spanning tree of $V(q)$ to g to reduce the combinations by the connectivity restriction.

QuickSI first finds a spanning tree T of the query q, and then convert q into a sequence $seq = [E[1], \ldots, E[|V(q)|]]$, called QI-Sequence. Each entry $E[i]$ has one and only one *spanning edge* $(E[i], E[j])$, denoted by $E[i].sEdge$, such that $j < i$ and $(E[i], E[j])$ is in T where $E[1].sEdge$ is the label of vertex $E[1]$. All other edges in q are called backward edges in seq and the set of backward edges *incident* to an entry $E[i]$ is denoted by $E[i].bEdges$.

To identify a subgraph-isomorphic mapping from q to g, QuickSI iteratively grows each possible mapping on T in a depth-first manner according to the vertices order in seq. QuickSI can terminate earlier if a prefix of seq cannot be sub-isomorphically mapped to g. To effectively reduce the search costs, QuickSI proposes to order the QI-Sequence seq as follows. Pick up the vertex v from q, such that its label has the *lowest* occurrence among the candidate graphs, as the 1st entry $E[1]$ in seq. Then, iteratively pick up an unchosen vertex as $E[i]$ (for $2 \leq i \leq |V(q)|$) such that the spanning edge has the lowest occurrence in the candidate graphs among all valid options.

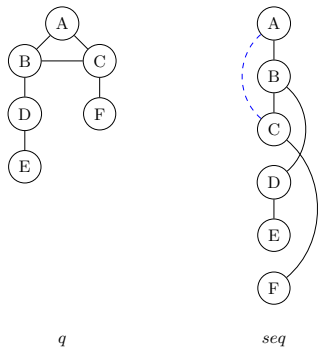


Figure 5: An Example Query and Its QI-Sequence

EXAMPLE 2. *A query q and its QI-Sequence are shown in Figure 5. The QI-Sequence has 6 entries. Spanning edges are depicted by solid lines and backward edges are depicted by dashed lines (only one in this example).*

3. DISTANCE BASED FILTERING

In this section, we first characterize a tight condition under which the triangular inequality holds. Then, we present the pruning and validation rules based on the triangular inequality. This is followed by the framework description.

3.1 Triangular Inequality

The triangular inequality regarding graph relaxation distances does not always hold. A counter example is given in Figure 6, where $dist(g_1, g_3) = 3$, $dist(g_1, g_2) = 0$, and $dist(g_2, g_3) = 1$. Below, we show that the triangular inequality holds under a *connectivity dominance* condition.

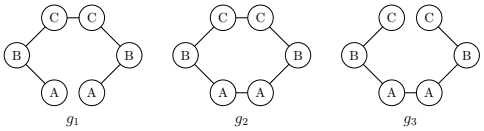


Figure 6: Counter Example

Definition 5. The connectivity of $mccs(g_1, g_2)$ *dominates* the connectivity of g_2 if there is a subgraph isomorphic mapping $\mathcal{F}$ from $mccs(g_1, g_2)$ to g_2 (i.e. $mccs(g_1, g_2) \subseteq_{\mathcal{F}} g_2$) such that if removing a set S of edges in $mccs(g_1, g_2)$ causes $mccs(g_1, g_2)$ disconnected, then removing $\mathcal{F}(S)$ in g_2 always causes g_2 disconnected.

In the above example, the connectivity of $mccs(g_1, g_2)$ does not dominates the connectivity of g_2 and the connectivity of $mccs(g_2, g_3)$ does not dominate g_2.

THEOREM 1. *Given three graphs g_1, g_2, and g_3, if the connectivity of $mccs(g_1, g_2)$ dominates the connectivity of g_2 or the connectivity of $mccs(g_3, g_2)$ dominates g_2, then $dist(g_1, g_3) \leq dist(g_1, g_2) + dist(g_2, g_3)$.*

PROOF. We first show that the theorem holds if the connectivity of $mccs(g_1, g_2)$ dominates the connectivity of g_2.

Suppose that $\mathcal{F}$ is a subgraph isomorphic mapping from $mccs(g_1, g_2)$ to g_2 such that if removing a set S of edges in $mccs(g_1, g_2)$ causes $mccs(g_1, g_2)$ disconnected, then removing $\mathcal{F}(S)$ in g_2 always causes g_2 disconnected. Note that $\mathcal{F}(mccs(g_1, g_2))$ and $mccs(g_2, g_3)$ are subgraphs of g_2, respectively. Below we first show that the common part of $\mathcal{F}(mccs(g_1, g_2))$ and $mccs(g_2, g_3)$ is either $\emptyset$ or a connected subgraph of g_2, denoted as $\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3))$.

Suppose that $\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3)$ ($\neq \emptyset$) is disconnected. Then, there are at least two connected components c_1 and c_2 in $\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3)$. Note that c_1 and c_2 are maximum in $\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3)$ and disconnected to each other. Let S' be the set of edges in $\mathcal{F}(mccs(g_1, g_2))$ each of which is either incident to a vertex in c_1 or to a vertex in c_2 but is not contained in c_1 or c_2. It is immediate that $S' \cap E(mccs(g_2, g_3)) = \emptyset$ since c_1 and c_2 are maximum in $\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3)$.

According to the definition of S', the removal of S' makes $\mathcal{F}(mccs(g_1, g_2))$ disconnected. Hence, the removal of $\mathcal{F}^{-1}(S')$ makes $mccs(g_1, g_2)$ disconnected. Therefore, the removal of S' makes g_2 disconnected according to the assumption; that is, $g_2 - S'$ is disconnected. Since $S' \cap E(mccs(g_2, g_3)) = \emptyset$, $c_1 \subset mccs(g_2, g_3)$, $c_2 \subset mccs(g_2, g_3)$, and $g_2 - S'$ is disconnected, it is immediate that $mccs(g_2, g_3)$ is disconnected. Contradicting! Therefore, $\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3)$ is either $\emptyset$ or connected. Thus,

$$|E(mccs(g_1, g_3))| \geq |E(\mathcal{F}(mccs(g_1, g_2))) \cap E(mccs(g_2, g_3)))| \quad (1)$$

We can represent $|E(g_2)|$ as follows where α (≥ 0) is the number of edges in g_2 not included in $\mathcal{F}(mccs(g_1, g_2))$ nor in $mccs(g_2, g_3)$.

$$\begin{aligned} |g_2| =& \alpha + |E(\mathcal{F}(mccs(g_1, g_2)))| + |E(mccs(g_2, g_3))| \\ & - |E(\mathcal{F}(mccs(g_1, g_2)) \cap mccs(g_2, g_3))| \end{aligned} \quad (2)$$

From (1) and (2), together with the definition of graph relaxation distance, the theorem follows.

Similarly, we can prove the theorem if the connectivity $mccs(g_3, g_2)$ dominates g_2. □

3.2 Pruning and Validation

Based on the triangular inequality, features can be used to filter non-promising graphs and to include (validate) graphs, with similarity guaranteed to exceed the given similarity threshold, into the answer set. Features discussed here could be any graph structures (paths, trees, subgraphs).

By Theorem 1, there could be totally 6 triangular inequalities among q, f, and g. It can be immediately shown that $dist(q, g) \leq dist(q, f) + dist(f, g)$ is equivalent to $dist(g, q) \leq dist(g, f) + dist(f, q)$, $dist(f, q) \leq dist(f, g) + dist(g, q)$ is equivalent to $dist(q, f) \leq dist(q, g) + dist(g, f)$, and $dist(f, g) \leq dist(f, q) + dist(q, g)$ is equivalent to $dist(g, f) \leq dist(g, q) + dist(q, f)$, respectively. Note that the equivalence of two inequalities also means that the connectivity dominance conditions to make the two inequalities hold are the same. Thus, there are essentially 3 different triangular inequalities among q, f, and g. We use two of them for pruning and one of validation.

As the verification of whether the connectivity of $mccs(q, g)$ dominates the connectivity of g involves computing $mccs(q, g)$, it does not make sense to use this condition in a pruning rule.

PRUNING RULE 1. *For a feature f, if the connectivity of $mccs(g, f)$ dominates the connectivity of g, then g can be pruned when $dist(q, f) - dist(g, f) > \sigma$.*

PROOF. Since the connectivity of $mccs(g, f)$ dominates the connectivity of g, $dist(q, f) \leq dist(q, g) + dist(g, f)$ according to Theorem 1. Thus, if $dist(q, f) - dist(g, f) > \sigma$, then $dist(q, g) > \sigma$. □

Similarly, $dist(f, g) \leq dist(f, q) + dist(q, g)$ gives the following pruning rule.

PRUNING RULE 2. *For a feature f, if the connectivity of $mccs(f, q)$ dominates the connectivity of q, then g can be pruned when $dist(f, g) - dist(f, q) > \sigma$.*

VALIDATION RULE 1. *For a feature f, if the connectivity of $mccs(f, q)$ dominates the connectivity of f or the connectivity of $mccs(f, g)$ dominates the connectivity of f, then g is a result graph when $dist(q, f) + dist(f, g) \leq \sigma$.*

PROOF. Note that $dist(q, g) \leq dist(q, f) + dist(f, g)$ holds according to Theorem 1. Thus, $dist(q, g) \leq \sigma$. □

3.3 Framework

Existing techniques [7, 4] follow the filtering-verification paradigm. In this paper, we propose an efficient algorithm DistVP that employs distances-based triangular inequalities for validation and pruning. It has three phases, pruning-validation-verification, based on our distance-based index, as shown in Figure 7. We outline the three phases of Algorithm DistVP as follows. Initially, put all data graphs g in C_q with $|E(q)| - |E(g)| \leq \sigma$.

1. **Pruning.** Regarding each indexed feature, a data graph will be removed from the candidate set C_q if the pruning conditions hold in Pruning Rules 1 or 2.
2. **Validation.** Graphs are immediately added to the result set V without an expensive verification if the conditions holds in Validation Rule 1.
3. **Threshold-based Verification.** The candidate graphs in $C_q - V$ are processed by our new threshold-based detection algorithms.

Next two sections will give the algorithmic details of Algorithm DistVP. Since our pruning, validation, and index construction techniques will use our verification technique. Next we first present our verification algorithm.

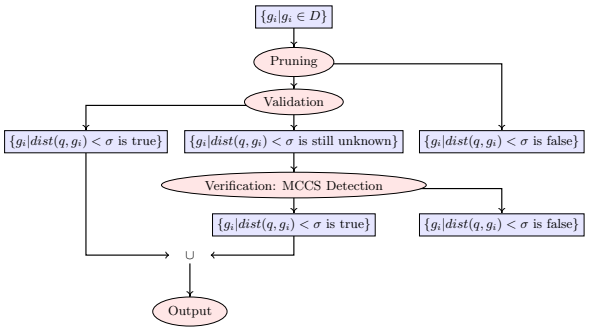


Figure 7: Pruning-Validation-Verification

4. EXPERIMENTS

Below is a summary of the techniques developed and implemented for a comprehensive performance study.

- **Verification**: There are no techniques available in the literature to compute MCCS-based similarity. We evaluate our verification algorithm based on two proposed strategies: Ad-HocStrategy and MemorizingStrategy; they are denoted by **AdHOC** and **MEMO**, respectively
- **Filtering:** We evaluate the pruning, validation, and GrafD-index techniques proposed.

We use the (only) two filtering algorithms in [7, 8] as the benchmark techniques to evaluate our techniques. We use **Grafil+** to denote the combination of Grafil filtering techniques [7] and our verification technique MEMO, use **editD** to denote the filtering technique in [8], and use DistVP to denote the combination of our filtering, pruning, and MEMO verification techniques. Since there is no code available for Grafil filtering techniques, we code them by ourself.

All algorithms are implemented in standard C++ with STL and complied with GNU GCC. Experiments were run on a PC with Intel Xeon 2.40GHz CPU and 4G memory running Debian Linux.

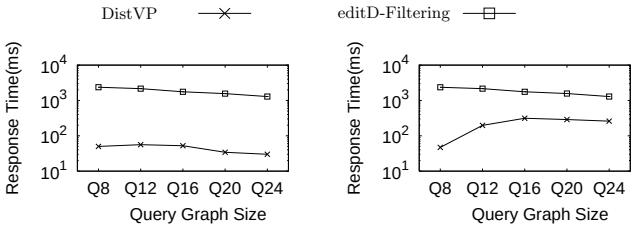
Real Datasets. A popular benchmark dataset, the AIDS antiviral database, is used in our performance evaluation. The dataset contains totally 62 distinct vertex labels. Following the recent performance study settings [1, 5, 6], edge labels are ignored for a *tough* evaluation. The default dataset consists of randomly chosen 10K graphs from AIDS. On average, each graph has 25.4 vertices and 27.3 edges.

Query Set. To thoroughly evaluate our techniques, we download the five benchmark query sets, $Q8$, $Q12$, $Q16$, $Q20$ and $Q24$ from the web-site as pointed and used by [1, 5, 6]. Each query graph in Q_i has exactly i edges.

Threshold in GrafD-index. The default value of the threshold k used in GrafD-index is 3.

Below we report the results of our performance study. Unless otherwise specified, we will use the above *default settings* in our experiment.

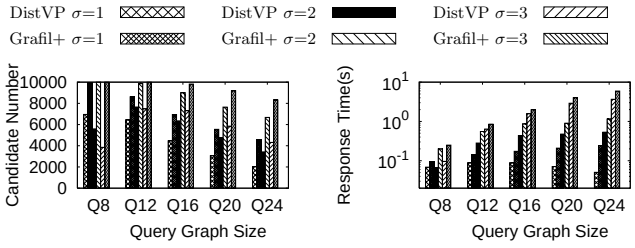
Evaluating Verification Techniques. Figure 8 reports the experiment results on the response time of our two verification algorithms, AdHOC and MEMO. The time recorded is the average response time per query. It shows that MEMO is significantly more efficient than AdHOC and can achive more than two orders of magnitude speed-up. Thus, in the rest of our experiment we use MEMO as the verification technique in DistVP and Grafil+.



(a) σ=1 (b) σ=2

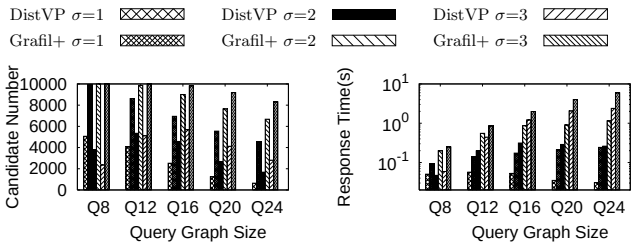
Figure 9: Comparing with editD

Comparing with editD. As depicted in Figure 9, the total computation time (pruning, validation, and verification) of DistVP is more efficient than the editD-based filtering technique in [8] when the similarity degree is high. Note that the released binary code by the authors of [8] outputs the filtering time only and does not provide the candidate graphs so that we cannot conduct the verification evaluation. On the other hand, the edit distance based filtering technique proposed in [8] is a general framework that serves for a wide range of graph structure search; it is unfair to continue to evaluate it only against the problem studied in the paper. These make us exclude the editD technique from a further evaluation.



(a) Candidate Size (b) Total Response Time

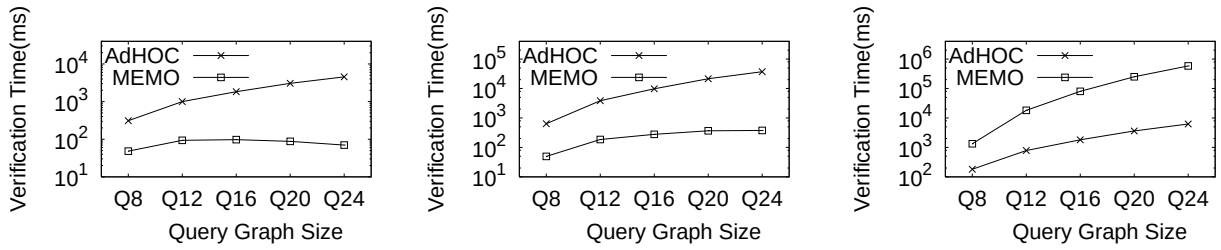
Figure 10: Using Grafil's features



(a) Candidate Size (b) Total Response Time

Figure 11: Using Our Features in DistVP

Comparing with Grafil+. In our first such experiment, we use the feature set selected by Grafil to compare our filtering (pruning and validation) techniques with Grafil. Figures 10(a) shows that the number of candidate graphs per query (on average) produced by our techniques (DistVP) is



(a) Query Relaxation $\sigma = 1$ (b) Query Relaxation $\sigma = 2$ (c) Query Relaxation $\sigma = 3$

Figure 8: Verification

significantly less than that by Grafil, about 20%–50% less. We further verify the effectiveness of our filtering techniques by recording the total response time per query on average. Since there is no existing verification technique, we use Grafil+ (Grafil +MEMO) for the purpose. Figures 10(b) shows that the total response time follows similar trends to those in Figures 10(a); this is mainly because the verification phase plays the dominant role. It is noteworthy that Grafil can hardly prune away data graphs when q is small and edge labels are removed. It is also noteworthy that the total response time increases significantly with query graph sizes; this is because the verification cost for large query graphs is much more expensive than the cost for small query graphs.

We further evaluate the effectiveness of our techniques by a set of features generated by the feature selection techniques with the frequency threshold 2% and discriminative ratio 2%. Then we compare with Grafil filtering techniques using its own features. As depicted in Figure 11, the number of candidate graphs generated by our techniques is significantly smaller comparing with the result in Figures 10(a). Now, the candidate set size by Grafil can be reduced up to 80%. In the rest of performance evaluation we will exclude Grafil and only focus on our techniques; the feature set in this experiment will be used thereafter.

5. CONCLUSION

In this paper, we investigate the problem of connected subgraph similarity search. We propose a filtering-*validation*-verification-based query processing framework with the aim to minimize the number of candidate graphs. A novel indexing technique, GrafD-index, is proposed which indexes data graphs based on defined distance functions. Effective and efficient pruning and validation techniques have been proposed based on GrafD-index. We also propose novel, efficient techniques to perform verification aiming to optimize the matching order and computational sharing. A comprehensive performance study against real datasets demonstrates that our filtering (pruning and validation) techniques are significantly outperform to the (only) two existing filtering techniques. Our techniques are also efficient and scalable.

As a possible future study, we will investigate the "optimal feature" selection problem if a query log exists, as well as this problem regarding the applications where graphs involved are larger, say, each graph has tens of thousands vertices.

6. REFERENCES

[1] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD*, pages 857–872, 2007.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[3] R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs. In *ICPR*, volume 2, pages 112–115 vol.2, 2002.

[4] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, pages 38–39, 2006.

[5] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. In *VLDB*, pages 364–375, 2008.

[6] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[7] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, pages 766–777, 2005.

[8] Z. Zeng, A. K. H. Tung, J. Wang, L. Zhou, and J. Feng. Comparing stars: On approximating graph edit distance. In *VLDB*, pages 25–36, 2009.

[9] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: tree + delta <= graph. In *VLDB*, pages 938–949, 2007.