Connected Substructure Similarity Search

Haichuan Shang[†‡] Xuemin Lin[†‡] Ying Zhang[†] Jeffrey Xu Yu[§] Wei Wang[†‡]

[†] University of New South Wales, Australia
[§] Chinese University of Hong Kong, China
[‡] NICTA, Australia
[†]{shangh, lxue, yingz, weiw}@cse.unsw.edu.au
[§]yu@se.cuhk.edu.hk

ABSTRACT

Substructure 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, GrafD-Index, to index graphs according to their "distances" to features. We characterize a tight condition under which the distance-based triangular inequality holds. We then develop lower and upper bounding techniques that exploit the GrafD-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.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Algorithms, Experimentation, Performance

Keywords

Graph Database, Similarity Search

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 [12, 15, 18, 22], structure-based graph queries [5, 6, 10, 11, 19, 23, 27, 26], etc.

Figure 1: Query Graphs

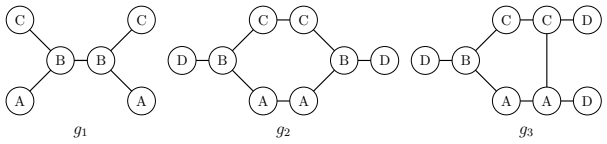


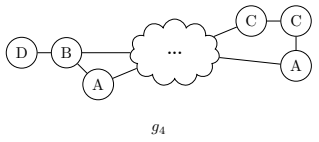
Figure 2: A Sample Graph Database

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 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, [24] 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 [24] 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

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

on the difference of the query graph and the MCS. It is well known that detecting MCS is NP-complete [9]. Hence, existing techniques [24, 25], 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).



g_4

Figure 3: Cloud Contains a Large Number of Nodes

The filtering techniques [24, 25] inherently do not provide 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 [24, 25] 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 [24, 25]. They also indicate that our total computation (filtering, validation, and verification) is more efficient than the filtering technique in [25] for high-similarity search. Our filtering and validation techniques significantly reduce (up to 80% size reduction) the size of the candidate set by Grafil [24]. 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 provides novel MCCS detection algorithms to verify the candidates. Section 5 presents GrafD-index, and pruning and validation algorithms based on GrafD-index. Section 6 reports the experimental results. The related work and conclusion are given in Section 7 and Section 8, respectively.

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

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

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 $|q| = |g|$. [24] 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.

2.2 Preliminaries

Grafil. Grafil [24] is developed to support efficient subgraph similarity searches and follows the *filtering-verification* query processing paradigm. It provides a feature-based index [10, 23] to effectively filter non-promising data graphs. Features could be paths [10], trees [27], or subgraphs [23].

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.

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.*

	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

QuickSI. An efficient verification algorithm, QuickSI [19], 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.

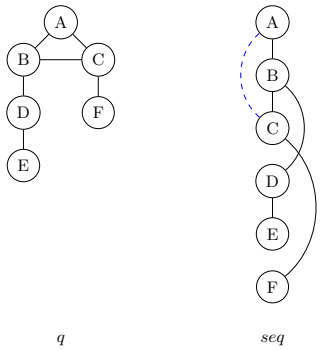


Figure 5: An Example Query and Its QI-Sequence

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.

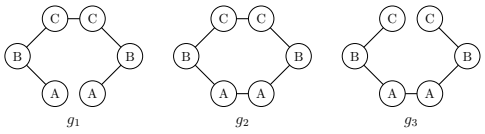


Figure 6: Counter Example

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. □

In Section 5.1, we will show that given an embedding (subgraph isomorphic mapping) $\mathcal{F}$ from $mccs(g_1, g_2)$ to g_2, it takes linear time to determine whether or not the conditions in Definition 5 are satisfied.

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, $dist(q,g) \leq \sigma$. □

In Section 5.3, we will present efficient techniques to implement these pruning and validation rules. Below we first present the framework of our approach.

3.3 Framework

Existing techniques [24, 11] 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 $|E(q)| - |E(g)| \leq \sigma$.

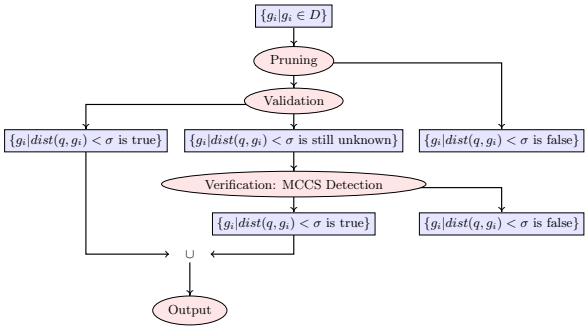


Figure 7: Pruning-Validation-Verification

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 in Section 4

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.

4. THRESHOLD-BASED VERIFICATION

In the verification phase, query answers are extracted from the candidate graphs, $C_q - V$. Given a q, a threshold σ on query relaxation distance, and a $g \in C_q - V$, we verify whether $dist(q,g) \leq \sigma$, instead of computing the exact value of $dist(q,g)$. Given a subgraph q' of q, a spanning subtree T of q', and a subgraph-isomorphic mapping $\mathcal{F}$ from T to g, let $L^{T,\mathcal{F}}$ denote the number of *mismatched* edges in q'; that is, edges $(u,v) \in q'$ but $(\mathcal{F}(u),\mathcal{F}(v)) \notin g$. Let $L^{q'} = |E(q) - E(q')|$. The theorem below is a key, which can be directly verified from the definitions.

THEOREM 2. *$dist(q,g) \leq \sigma$ if and only if there is a spanning tree T of a connected subgraph q' of q and a subgraph isomorphic mapping $\mathcal{F}$ from T to g such that*

$$L^{T,\mathcal{F}} + L^{q'} \leq \sigma \tag{3}$$

A trivial way for verification is to enumerate all feasible sub-spanning trees of q and then use QuickSI to explore them one by one to determine if we can find a mapping such that the inequality in (3) holds. Nevertheless, this misses the opportunity of sharing the computation among different sub-spanning trees with a common part and may unnecessarily enumerate too many different spanning tress.

To resolve the above issues, the central idea of our algorithm is to enumerate an alternative sub-spanning tree of q based on demands and only when it is feasible to extend the current partial mapping. In our algorithm, enumerating sub-spanning trees of q is conducted in a depth-first search manner from a QI-Sequence seq combining with QuickSI to explore possible mapping on current spanning tree T. A key issue is that the following may happen. The current partial mapping on T cannot be extended further with the requirement to exactly map the current spanning edge in T to g; for instance, regarding Example 2 the spanning edge (B,C) in the QI-Sequence in Figure 5 simply cannot be mapped to an edge in g restricted to the current partial mapping. Hence, to **ensure the connectivity** we need to find a backward edge in seq to replace (B,C) to form a new QI-Sequence to continue the subsequent search using QuickSI.

For the simplicity of presentation, below we first present our algorithm with a special requirement that the first entry of a QI-Sequence of q has to be exactly mapped to g in computing $dist(q,g)$. Algorithm 1 below outlines our threshold-based algorithm that iteratively extends the current partial mapping on $seq[1,...,l-1]$ in a depth-first search manner, where $seq[1,...,l]$ denotes the subgraph of q consisting of the first l entries and the (spanning and backward) edges among these l entries in seq. It terminates whenever **true** returns and returns **false** if any mapping with the 1st entry to be exactly mapped to g has more than σ edges miss-matched.

Algorithm 1: **FixHeadVerify**(σ, g, seq, l)

```
   Input  : σ is a given query relaxation threshold;
            g is a data graph;
            seq is q's QI-Sequence;
            l is the current depth, initially 1;
1  if UnmatchedEdges(seq, l) ≤ σ then return true ;
2  while NextMatch(E[l].sEdge, g) do
3      ε ← MissingBackwardEdges(E[l].bEdges, g);
4      if ε ≤ σ then
5          if FixHeadVerify(σ − ε, g, seq, l + 1) then
6              return true
7  if σ ≥ 1 & l ≥ 2 then
8      seq' ← GetNewSeq(seq − E[l].sEdge);
9      if |seq'| ≥ l then
10         ε ← MissingEdges(seq − E[l].sEdge);
11         if ε ≤ σ then
12             if FixHeadVerify(σ − ε, g, seq', l) then
13                 return true
14 return false
```

Details of Algorithm 1. Lines 1 exams whether or not Algorithm 1 can terminate in the current round with a posi-

tive answer, **true**, where UnmatchedEdges($E[l].sEdge, l$) calculates $L^{T,\mathcal{F}} + L^{q'}$ in (3) based on the current subgraph isomorphic mapping $\mathcal{F}$ on the spanning tree T of $seq[1, ..., l-1]$.

Lines 2-6: Determine if the current partial subgraph isomorphic mapping from the spanning tree T of $seq[1, ..., l-1]$, allowing at most σ edges in $seq[1, ..., l-1]$ mismatched, can be extended to $E[l].sEdge$ and beyond while still allowing at most σ edges mismatched. Recall, as defined in Section 2.2 $E[l].sEdge$ ($l \geq 2$) is the spanning edge between an entry in $seq[1, ..., l-1]$ to $E[l]$ and $E[1].sEdge$ is the label of $E[1]$.

NextMatch($seq[l].sEdge, g$) maps $E[l].sEdge$ to an edge in g while preserving the current mapping on the spanning tree of $seq[1, ..., l-1]$. Note that there could be more than one edge in g to be mapped from $E[l].sEdge$. Due to the connectivity requirement, we only need to check those edges *incident* to the new vertex brought by $E[l].sEdge$. The number of backward edges from the entry $E[l]$ to entries in $seq[1, ..., l-1]$, not matched by the current mapping, is calculated by MissingBackwardEdges($E[l].bEdges, g$). Note that we need to consider all possible mappings of the current spanning edge to g and this is achieved by the **while** loop in Line 2. Here, $\epsilon > \sigma$ (checked by line 4) means that in the current $\mathcal{F}$ on the spanning tree T of $seq[1, ..., l]$, $L^{T,\mathcal{F}} > \sigma$; consequently, the current $\mathcal{F}$ cannot be extended further.

line 7: checks whether or not the current mapping on a spanning tree T of $seq[1, ..., l-1]$ can be extended, given $E[l].sEdge$ cannot be used in the mapping. Note that $l = 1$ means that at this point any mapping with an exact mapping on $E[1]$ already has been shown to violate the inequality (3); thus, Algorithm 1 returns **false** and terminates.

Lines 8-13: tries to find possible extensions to the current mapping on $seq[1, ..., l-1]$, given that at this point, the algorithm already concludes any mapping restricted to the current mapping on $seq[1, ..., l-1]$ with an exact mapping on $E[l].sEdge$ always violates the inequality (3). Thus, $E[l].sEdges$ needs to be removed from the spanning tree T of seq.

Once $E[l].sEdge$ is removed from T, T is cut into two sub-trees: T_1 contains the entries in $seq[1, ..., l-1]$ and T_2 contains $E[l]$. GetNewSeq($seq - E[l].sEdge$) first finds out if T_1 and T_2 can be connected by a backward edge in seq. If yes, then a backward edge is used as a new spanning edge to connect T_1 and T_2 to form a new spanning tree for ($seq - E[l].sEdge$) that gives a new QI-Sequence seq'; otherwise T_1 is used to give a new QI-Sequence seq'. Noth that in both cases, the first $(l-1)$ entries in seq' are the same as those in seq.

If $|seq'| = l - 1$ (i.e., impossible to extend $seq[1, ..., l-1]$ due to the cut of $E[l].sEdge$) then the current round should terminate because the test in line 1 fails. Otherwise, MissingEdges($seq - E[l].sEdge$) gives $\epsilon = 1$ if T_1 and T_2 can be connected by a backward edge or $\epsilon = m$ if seq' is obtained on T_1 where m is the number of edges in seq incident to a vertex (entry) in T_2.

Conducting GetNewSeq(). There two possible strategies to conduct GetNewSeq($seq - seq[l].sEdge$).

1. *Ad-HocStrategy*: Each time, conduct GetNewSeq($seq - E[l].sEdge$) on the fly such that if T_1 and T_2 can be connected by a backward edge, then a backward edge with the minimum frequency among all valid edges is chosen; otherwise discard T_2.

2. *MemorizingStrategy*: GetNewSeq($seq - E[l].sEdge$) includes to determine T_1 and T_2, to find out if such a backward edge exists (then get such a backward edge if yes), and to order entries other than the first $(l-1)$ entries in seq'. Hence, a potential problem with Ad-HocStrategy is that the computation costs incur every time. Observe that if seq and $E[l].sEdge$ are the same, then the result of GetNewSeq($seq - E[l].sEdge$) is always the same. MemorizingStrategy memorizes all the previous results of GetNewSeq($seq - E[l].sEdge$) so that GetNewSeq($seq - E[l].sEdge$) conducts Ad-HocStrategy only when the result is unavailable. Consequently, the costs of re-processing GetNewSeq($seq - E[l].sEdge$) can be saved. Below are the implementation details.

A binary tree $\mathcal{T}$ is kept in the buffer, called buffer tree, to memorize all the results of GetNewSeq() encountered so far, as well as to guide the search. Each node in $\mathcal{T}$ represents a spanning edge in a QI-Sequence, while the root of $\mathcal{T}$ represents $E[1].sEdge$ in the QI-Sequence of q. Iteratively, if the spanning edge at a node t of $\mathcal{T}$ has a match (line 2) and the threshold constraint holds (line 4), then Algorithm 1 visits the left child of t; otherwise, Algorithm 1 visits the right child of t as follows if the conditions in line 7 holds. We iteratively grow $\mathcal{T}$ as follows. The spanning edges of the obtained QI-Sequence of q are loaded in $\mathcal{T}$ as the left-most branch according to the order of entries. Iteratively, Algorithm 1 starts from the root $E[1].sEdge$. When the current entry's spanning edge (except $E[1].sEdge$) has to be discarded, line 8 in Algorithm 1 will create the right-child $seq'[l].sEdge$ of the current entry $E[l].sEdge$ and load in spanning edges of the subsequent QI-Sequence (i.e $seq'[l, ...,]$) as the left-most branch of the sub-binary tree with $seq'[l].sEdge$ as the root if the right child does not exist. Here, $seq'[l]$ is the lth entry of seq'. Algorithm 1 may continue to drill down to the left child of $seq'[l].sEdge$ if it has a match and the threshold constraints hold (line 4). Recursively tracing back in Algorithm 1 follows exactly the same strategy of tracing back in a depth-first search on $\mathcal{T}$. Note that if it traces back to the root $E[1].sEdge$ of $\mathcal{T}$ and the root does not have a new matching in g, then Algorithm 1 terminates and returns **false**.

Illustrating Algorithm 1. A data graph g and the QI-Sequence seq of q are shown in Figure 8 (a) and (b), respectively, where dashed-lines depict the backward edges. Assume that $\sigma = 3$.

Suppose that Algorithm 1 iteratively drills down and maps the spanning edges (A, B) and (B, C) in seq to the *left* (A, B) and (B, C) of g, respectively, where edges connecting shaded nodes in Figure 8 (a) illustrate the current mapping. Due to the connectivity restriction, the spanning edge (B, D) in seq cannot be mapped into g; consequently, (B, E) or (B, F) has to be chosen by line 8 to continue the drilling-down. Assuming (B, E) is chosen as a new spanning edge after removing (B, D) in this round, then the new QI-Sequence (generated by line 8) is depicted in Figure 8 (c) where double lines show the current mapping to be retained. Similarly, since no match in g can be found for (B, E) due to the connectivity requirement, line 8 generates another QI-Sequence in Figure 8 (d). Now, due to missing-matching for (B, F) in the QI-Sequence, line 8 generates QI-Sequence in Figure 8 (e). Line 9 then excludes a further extension of the mapping on (B, C) and (A, B).

Therefore, Algorithm 1 recursively traces back to the state

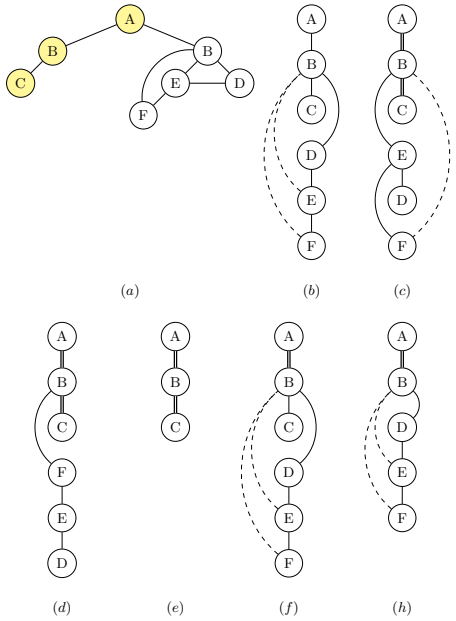


Figure 8: Verification Example ($\sigma = 3$)

in Figure 8 (f) and try to find another mapping from (B, C) in q to g. Since there is no other matching in g for (B, C) in q, (B, C) is removed from the current drilling-down. Line 8 generates the resultant QI-Sequence as depicted in Figure 8 (h). Similar to the above, it is found impossible to have $dist(q, g) \leq 3$ while retaining the current mapping on (A, B).

Algorithm 1 then traces back and tries to find another mapping from (A, B) in seq in 8(b) to g. Since such a mapping exists (i.e., the 2nd (A, B) in g), Algorithm 1 iteratively drills down starting from seq in Figure 8 (f). This time, due to connectivity, no match exists in g for (B, C) in seq. Consequently, line 8 again generates the QI-Sequence in Figure 8 (h). We eventually found a sub-isomorphic mapping from QI-Sequence in Figure 8 (h) to g and only one edge (B, C) in q does not have a match. Thus, Algorithm 1 returns **true**. □

The above example also shows that even the same selection of a QI-Sequence, Figure 8(h), may occur multiple times in processing one data graph. Therefore, MemorizingStrategy does save the costs of re-computation.

General Algorithm. Algorithm 1 can only deal with the situation where the first vertex in seq of q has an exact matching in computing $dist(q, g)$. Nevertheless, it can be immediately extended to a general case. Algorithm 2 below is a brief description where $K = |E(q)|$.

In Algorithm 2, $E[1]$ is the first entry of seq - a vertex in q. Decompose $(seq - E[1])$ is to decompose the connected components of $(q - E[1])$ and then get a QI-Sequence for each connected component; SEQ_{sub} consists of all such QI-Sequences. Note that if $(K - |E(seq_s)|) > \sigma$, then $|E(q)| - |E(mccs(q_s, g))| > \sigma$ where q_s is the graph which is represented by seq_s; thus, we do not need to consider q_s. In our implementation, we only generate QI-Sequence satisfying the condition in line 6 and load them in SEQ_{sub} to save the costs to generate useless QI-Sequence; that is, line 6 is used in Decompse ().

Correctness and Complexity. Based on Theorem 2, it is immediate that Algorithm 1 and Algorithm 2 are correct.

Algorithm 2: **Verify**(K, σ, g, seq)

```
   Input  : seq: QI-Sequence of q;
            g: data graph;
            σ: the query relaxation threshold;
1  if FixHeadVerify(σ, g, seq, 1) then
2  |  return true
3  else
4  |  SEQ_sub := Decompose (seq − E[1]);
5  |  for each seq_s ∈ SEQ_sub do
6  |  |  if (K − |E(seq_s)|) ≤ σ then
7  |  |  |  if Verify(K, σ − (K − |E(seq_s)|), g, seq_s) then
8  |  |  |  |  return true
9  |  return false
```

Using MemorizingStrategy, the space complexity required for the buffer tree is as follows.

THEOREM 3. *The total number of nodes in the tree T is $O(|E(q)|^{\sigma+1})$.*

PROOF SKETCH. Clearly, each node in the tree corresponds to a state that the mapping on the its ancestors with at most σ spanning edges are missed. Therefore, the total number of nodes is at most $\sum_{i=0}^{\sigma} \binom{|E(q)|}{i} = O(|E(q)|^{\sigma+1})$. □

The total buffer tree size is bounded by $O(|E(q)|^{\sigma+2})$ since each subsequent QI-Sequence takes $O(|E(q)|)$ space.

GetNewSeq involves two kinds of computation: 1) determine a backward edge to connect two sub-spanning trees, and 2) re-organize the subsequent QI-Sequence. This can be done $O(|E(q)| \log |E(q)|)$. Note that Algorithm 1 is similar to QuickSI if only one QI-Sequence is involved and equivalent to QuickSI if $\sigma = 0$. Therefore, using MemorizingStrategy the time complexity of Algorithm 1 runs in time $O(|E(q)|^{\sigma+1}(|E(q)| \log |E(q)| + C_{\text{QuickSI}}))$ for each data graph g in the worst case where C_{QuickSI} is the time of QuickSI. Although still exponential in the worst case due to the NP-completeness, our experiments demonstrate it is very efficient in practice.

5. FILTERING TECHNIQUES

In this section we present an inverted indexing technique, GrafD-index, to index graphs g based on $dist(f, g)$ for each feature f. Then, we present efficient techniques to conduct pruning and validation. Finally, we discuss feature selections. Below, we first present our techniques to compute MCCS and to determine connectivity dominance.

5.1 MCCS, *dist*, and Connectivity Dominance

MCCS and *dist*. Given two graphs g_1 and g_2, Algorithm 2 can return a common graph $g_{1,2}$ of g_1 and g_2 with $|E(g_1)| - |E(g_{1,2})| \leq k$ if $dist(g_1, g_2) \leq k$, g_1 is used as q, g_2 is used as g, and k is used as the distance threshold in the algorithm. Although there is no guarantee that $g_{1,2}$ is $mccs(g_1, g_2)$, Algorithm 2 can be immediately modified as follows to generate $mccs(g_1, g_2)$ if $dist(g_1, g_2) \leq k$.

When Algorithm 2 returns **true**, we output g_c, consisting of the vertices in the current sub-spanning tree T of g_1 and the edges (in g_1) each of which is incident to two vertices in T, as a candidate of $mccs(g_1, g_2)$. Instead of termination when **true** returns, we continue Algorithm 2 by iteratively replacing the candidate of $mccs(g_1, g_2)$ with the current g_c and changing the distance threshold to $(|E(g_c)| - 1)$ till **false**

returns or **true** returns with the distance threshold 0. The last kept g_c is $mccs(g_1, g_2)$.

Clearly, $mccs(g_1, g_2)$ can also be calculated if $dist(g_2, g_1) \leq k$. Theorem 4 below implies that we only need to explore the smaller graph between g_1 and g_2, as q in Algorithm 2, to compute $mccs(g_1, g_2)$ when $\min\{dist(g_1, g_2), dist(g_2, g_1)\} \leq k$. Consequently, $dist(g_1, g_2)$, and $dist(g_2, g_1)$ can also be exactly computed under this situation.

THEOREM 4. *If $|E(g_1)| \leq |E(g_2)|$, then $dist(g_1, g_2) \leq dist(g_2, g_1)$.*

PROOF. Based on the fact that $dist(g_1, g_2) = dist(g_2, g_1) - |g_2| + |g_1|$, the theorem holds. □

Connectivity Dominance Testing. The theorem below is a key to the connectivity domination test.

THEOREM 5. *The connectivity $mccs(g_1, g_2)$ dominates the connectivity of g_2 if and only if there is an sub-isomorphic mapping $\mathcal{F}$ from $mccs(g_1, g_2)$ to g_2, such that none of two vertices of $F(mccs(g_1, g_2))$ belong to the same connected component of $g_1 - E(\mathcal{F}(mccs(g_1, g_2)))$.*

PROOF. The following can be immediately verified. Given a subgraph isomorphic mapping $\mathcal{F}$ from $mccs(g_1, g_2)$ to g_2, $\exists\ S' \subseteq E(mccs(g_1, g_2))$ such that the removal of S' causes $mccs(g_1, g_2)$ disconnected but the removal of $\mathcal{F}(S')$ still leaves g_2 connected if and only if $\exists$ two vertices ($\in \mathcal{F}(mccs(g_1, g_2))$) in the same connected component of $g_2 - E(\mathcal{F}(mccs(g_1, g_2)))$. Thus, the theorem holds according to the definition of connectivity dominance. □

Our algorithm for connectivity dominance test is outlined below in Algorithm 3; it terminates once **true** returns.

Algorithm 3: **C-Domination** ($mccs(g_1, g_2)$, g_1)

```
   Output : B: True or Fase
1  B := false;
2  for each embedding F(mccs(g1, g2)) do
3      if no 2 vertices in F(mccs(g1, g2)) fall in one
       connected component of g1 − E(F(mccs(g1, g2))) then
4          B := True;
```

Theorem 5 guarantees that Algorithm 3 is correct. In our implementation, we extend QuickSI [19] to enumerate all sub-isomorphic mappings F from $mccs(g_1, g_2)$ to g_1 by continuing QuickSI to get all mappings instead of the termination when **true** returns; that is, the modified QuickSI terminates only when **false** returns. Once an embedding $\mathcal{F}(mccs(g_1, g_2))$ is obtained, line 3 in Algorithm 3 can be conducted in a linear time by applying the computation of connected component.

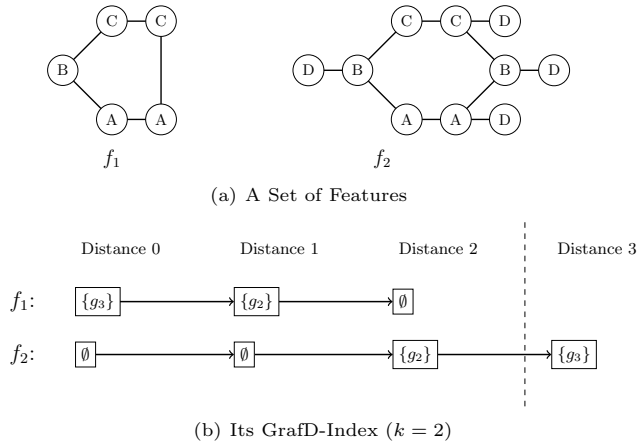
5.2 GrafD-Index

In GrafD-index, for efficiency reason, we use a distance threshold k. That is, for $\min\{dist(g, f), dist(f, g)\} \leq k$ we compute $mccs$ and $dist$ between g and f by the techniques described in Section 5.1. For each feature f, graphs g with exact values of $dist(f, g)$ are indexed according to an increasing order of $dist(f, g)$. Let $list_{f,j}$ denote the set of graphs with $dist(f, g) = j$, while $list_{f,\infty}$ is used to load in the set of graphs without exact values. In GrafD-index, for each feature f in $list_{f,j}$ ($\forall$ finite j) we only keep graphIDs and indicate if the connectivity of $mccs(f, g)$ dominates the connectivity of f (Pruning Rule 1) or dominates the connectivity of g (Validation Rule 1) or neither.

REMARK 1. *It is possible that the connectivity of $mccs(f, g)$ dominates both the connectivity g and the connectivity of f; for instance $f = g$.*

According to the discussions in Section 5.1, $dist(f, g)$ is calculated from $(dist(g, f) + |E(f)| - |E(g)|)$ if $E(f) > E(g)$. Thus, the exact value of $dist(f, g)$ may be larger than k though $\min\{dist(f, g), dist(g, f)\} \leq k$.

EXAMPLE 3. *Consider the data graphs and features in Figures 2 and 9(a), respectively. Assume that $k = 2$. Figure 9(b) shows the graphs indexed with the exact values of $dist(f_i, g)$ $(i = 1, 2)$. Note that the exact value of $dist(f_2, g_3)$ $(> k)$ is obtained because $dist(g_3, f_2)$ $(= 1)$ can be exactly computed. $dist(f_1, g_1)$ and $dist(f_2, g_1)$ cannot be exactly computed, and thus put in $list_{f_1,\infty}$ and $list_{f_2,\infty}$, respectively.*



(a) A Set of Features

(b) Its GrafD-Index ($k = 2$)

Figure 9: A GrafD-Index Example

5.3 Pruning and Validation

The filtering is based on Pruning Rules 1 and 2, as well as Validation Rule 1. In the filtering phase, we also use k as the distance threshold to compute $dist(f, q)$, $dist(q, f)$, and $mccs(f, q)$. We initially put all graphs g with $|E(q)| - |E(g)| \leq \sigma$ to the candidate set C_q and set $V = \emptyset$. The filtering phase proceeds as follows. Iterate over each feature f, we calculate $mccs(q, f)$, $dist(q, f)$ and $dist(f, q)$, as discussed above in Section 5.1 regarding k, and:

- **Pruning Rule 1** is firstly used to remove graphs g from C_q such that the connectivity of $mccs(g, f)$ dominates the connectivity of g and $dist(q, f) - dist(g, f) > \sigma$.
- **Pruning Rule 2** is then used to remove graphs from C_q such that $dist(f, g) - dist(f, q) > \sigma$ and the connectivity of $mccs(f, q)$ dominates the connectivity of q
- **Validation Rule 1** is finally used as follows. If the connectivity of $mccs(f, q)$ dominates the connectivity of f or the connectivity of $mccs(g, f)$ dominates the connectivity of f, then move the graphs g from C_q to V when $dist(q, f) + dist(f, g) \leq \sigma$.

Notes. With $dist(f,q)$, we also get $mccs(f,q)$, $mccs(q,f)$ by the obtained subgraph isomorphic mapping from $mccs(f,q)$ to q, and $dist(q,f)$ if $\min\{dist(q,f), dist(f,q)\} \leq k$. If $\min\{dist(q,f), dist(f,q)\} > k$, then none of them can be computed. Consequently, Pruning Rule 2 cannot be used since we are unable to check connectivity dominance, and Validation Rule 1 cannot be used since σ used is always not greater than k ($\sigma \leq k$); in Pruning Rule 1, we set $dist(q,f) = (k+1)$ to remove data graphs g with exact values of $dist(g,f)$ such that the pruning rule holds.

For data graphs g in $list_{f,\infty}$, $mccs(g,f)$ and exact values of $dist(f,g)$ ($> k$) and $dist(g,f)$ ($> k$) are unable to be determined. Similarly, Pruning Rule 1 and Validation Rule 1 cannot be used; in Pruning Rule 2, $(k+1)$ is used to represent $dist(f,g)$.

Finally, we conduct Pruning Rule 1 first because the connectivity dominance is pre-computed, and thus it is cheaper than conducting Pruning Rule 2.

5.4 Feature Selection

Theorem 5 implies that the connectivity dominance conditions specified in our pruning rules require that $mccs(f,g)$ and $mccs(f,q)$ are at least *induced* subgraphs [2] of g and q, respectively. As there is no knowledge towards query graphs, we can only impose the requirement on $mccs(f,g)$ and g. There are two ways to make $mccs(f,g)$ be an induced subgraph of g: 1) f is an induced subgraph of g, or 2) g is a subgraph of f. We use the first way since using super-graphs of data graphs as features is not cost effective and may make filtering costs more expensive than the costs of directly conducting verification. Consequently, we require features to be induced subgraphs of (some) data graphs; this will also make selected features have high-connectivity and thus increase the chances for the connectivity of $mccs(f,q)$ to dominate the connectivity of q.

Similarly, the connectivity dominance conditions in Validation Rule 1 require that $mccs(f,g)$ is at least an induced subgraph of f, or $mccs(f,q)$ is at least an induced subgraph of q. Again, since we only have knowledge about data graphs, we can only impose the requirement on g and f. In this case, two ways to make $mccs(f,g)$ be an induced subgraph of f are: 1) g is an induced subgraph of f, or 2) f is a subgraph of g. Similarly, this time we choose to use the second way; that is, require features to be subgraphs of (some) data graphs. Note that here, there is no requirement in the validation rule for features to be induced subgraphs.

Discriminative Frequent Sub-Induced Graphs. From the above two observations, good features should be induced subgraphs of (some) data graphs. As observed in [23], selecting a large number of features may reduce the number of candidate graphs but will also increase the filtering costs. Therefore, it is desirable to select representative induced subgraphs from data graphs as features. We extend gSpan algorithm [22] to mine *discriminative* [23] frequent induced subgraphs from data graphs as features.

Frequent Large Sparse Subgraphs. A potential problem with the above feature selection is that sizes of obtained features may not be large enough to use Validation Rule 1 against large query graphs. For instance, if the size of q is significantly larger than the maximum size of f, then Validation Rule 1 does not work because $dist(q,f)$ may be already greater than σ.

To resolve this, we mine discriminative frequent subgraphs with size greater than m_f from the set D_{m_f} such that the size of each graph in D_{m_f} is larger than m_f where m_f is the maximum size of already selected features (i.e., discriminative frequent sub-induced graphs).

For efficiency reason, in our implementation we first randomly choose some subtrees from graphs in D_{m_f} as seeds; the number of randomly selected seeds is $\frac{1}{3}$ of the number of already selected features (i.e., discriminative frequent sub-induced graphs) and each seed has size (m_f+1). The factor $\frac{1}{3}$ is used because there are 2 pruning rules and 1 validation rule; and we want to balance the costs used for validation in the filtering processing. Then we use gSpan algorithm to mine discriminative frequent subgraphs from those randomly selected seeds. Again, for efficiency reasons, gSpan algorithm is modified as follows. Let F_i denote the generated features with size i from D_{m_f} and $D_{m_f,i}$ denote the set of data graphs in D_{m_f} that contain at least one feature in F_i. In the modified gSpan algorithm, we first choose the minimum subset F_i' of F_i such that each data graph in $D_{m_f,i}$ contains at least one feature in F_i'. Then, the next iteration of the modified gSpan starts from F_i' instead of F_i. Note that the problem of finding F_i' is NP-hard since it is equivalent to the *minimum set cover* problem [9]; we use the greedy algorithm to obtain F_i' from F_i.

The union of features obtained by the above two techniques gives the features to our GrafD-index. Note that in applications where query patterns are collected and useful, learning techniques may be developed to choose the most effective features. Due to space limits, this will not be studied in the paper.

6. EXPERIMENTS

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

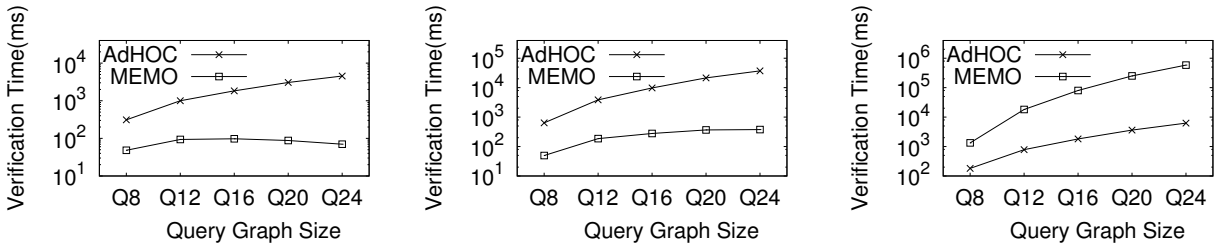
- **Verification**: There are no techniques available in the literature to compute MCCS-based similarity. We evaluate our verification algorithm based on two proposed strategies: Ad-HocStrategy and MemorizingStrategy; they are denoted by **AdHOC** and **MEMO**, respectively
- **Filtering:** We evaluate the pruning, validation, and GrafD-index techniques proposed in Section 3 and Section 5.

We use the (only) two filtering algorithms in [24, 25] as the benchmark techniques to evaluate our techniques. We use **Grafil+** to denote the combination of Grafil filtering techniques [24] and our verification technique MEMO, use **editD** to denote the filtering technique in [25], 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

[2] A subgraph g' of g is induced if any edge in g connecting 2 vertices in g' is also in g'.



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

Figure 10: Verification

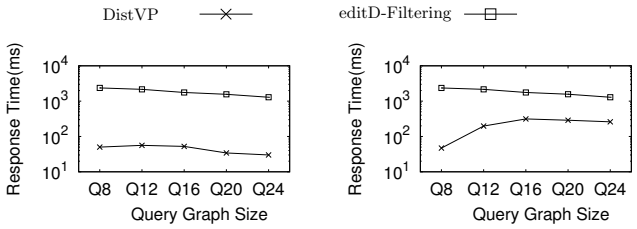
dataset contains totally 62 distinct vertex labels. Following the recent performance study settings [6, 19, 23], edge labels are ignored for a *tough* evaluation. The default dataset consists of randomly chosen 10K graphs from AIDS. On average, each graph has 25.4 vertices and 27.3 edges.

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 [6, 19, 23]. 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.

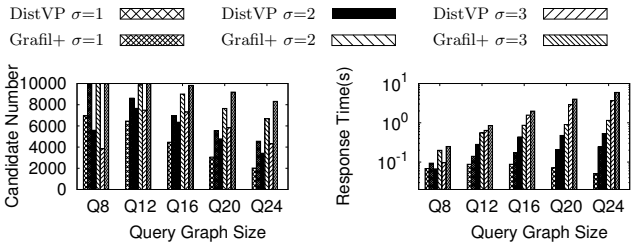
Evaluating Verification Techniques. Figure 10 reports the experiment results on the response time of our two verification algorithms, AdHOC and MEMO. The time recorded is the average response time per query. It shows that MEMO is significantly more efficient than AdHOC and can achive more than two orders 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

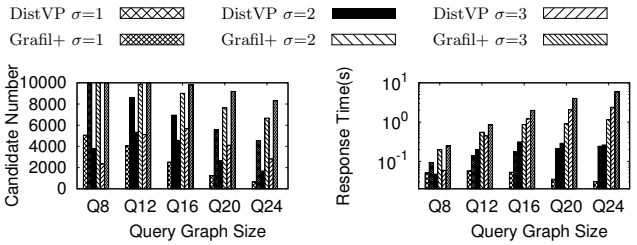
Figure 11: Comparing with editD

Comparing with editD. As depicted in Figure 11, the total computation time (pruning, validation, and verification) of DistVP is more efficient than the editD-based filtering technique in [25] when the similarity degree is high. Note that the released binary code by the authors of [25] 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 [25] 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 12: Using Grafil's features



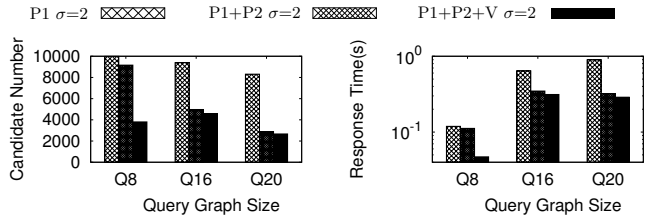
(a) Candidate Size (b) Total Response Time

Figure 13: 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 12(a) shows that the number of candidate graphs per query (on average) produced by our techniques (DistVP) is 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 12(b) shows that the total response time follows similar trends to those in Figures 12(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 in Section 5.4, with the frequency threshold 2% and discriminative ratio 2%. Then we compare with Grafil filtering techniques using its own features. As depicted in

Figure 13, the number of candidate graphs generated by our techniques is significantly smaller comparing with the result in Figures 12(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.



(a) Candidate Size (b) Total Response Time

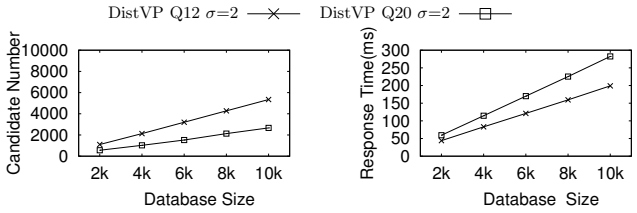
Figure 14: Filtering Power

Evaluating Filtering Power. We use P1 and P2 to represent the Pruning Rules 1 and 2, respectively; V is used to represent Validation Rule 1. As depicted by Figure 14, P1 and P2 both favor large query graphs while V favor small query graphs.

	Q8	Q12	Q16	Q20	Q24
Filtering (ms)	1.9	2.9	5.3	10.1	19.1
Verification (ms)	45.2	194.8	308.0	278.1	241.9

Figure 15: Filtering vs. Verification ($\sigma = 2$)

Filtering vs. Verification We evaluate the filtering time against the verification time where the time recorded is the average time per query and $\sigma = 2$ is used. As depicted by Figure 15, verification time plays the dominant role in our query processing.

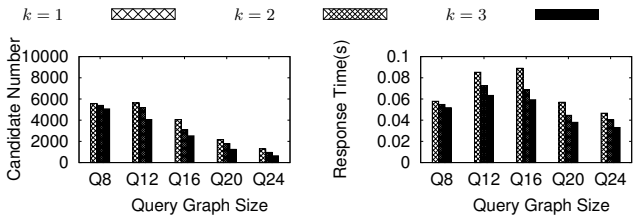


(a) Candidate Size (b) Total Response Time

Figure 16: Scalability

Scalability. We evaluate the scalability of our techniques regarding pruning powers and total response time using different database sizes with $\sigma = 2$. Here, data graphs in each database are randomly selected from the real dataset. We plot the candidate size and the total response times in Figure 16. The result shows our algorithm is scalable to the sizes of graph databases.

Impacts of Threshold (k). We evaluate the impact of distance threshold k (used in GrafD-index) on our technique with $\sigma = 1$. k varies from 1 to 3. The experiment results in Figure 17 show, as expected, that the GrafD-index is more effective when k increases.



(a) Candidate Size (b) Average Response Time

Figure 17: Various k

Index Construction Cost. We also evaluate the GrafD-index construction costs. The running costs are 932s for building our GrafD-index with $k = 1$, 1486s for $k = 2$, and 2018s for $k = 3$ by excluding the time of mining features.

Summary. Our experiment demonstrates that the techniques developed are efficient, effective, and scalable against sparse graphs. They outperform the existing filtering techniques in [24, 25]. Our new verification algorithm MEMO is both very efficient and scalable against real datasets.

7. RELATED WORK

Research in substructure search has attracted a great deal of attention from database research community. Due to NP-completeness, techniques for processing a large number of data graphs follow the filtering-verification paradigm. Most work in subgraph containment search focuses on developing effective and efficient indexing techniques, including, GraphGrep [10], gIndex [23], Closure-Tree [11], TreePi [26], gString [13], a graph decomposition based index [21], FG-Index [6], Tree+δ [27], GCoding [28], etc. In [5], cIndex is developed for the reverse version of subgraph containment search, namely *super-graph containment search* .

Recently, graph structure-based similarity search has also attracted considerable attention. In [11], Closure-Tree techniques have been extended to identify the K data graphs which are the most nearly isomorphic to the query graph; that is, full-graph similarity search. In [21], a variation of substructure similarity search, with a stronger constraint, is proposed. It aims to find data graphs with the minimum number of miss-matchings of vertex and edge labels bounded by a given threshold and disallows the size of a query graph greater than that of data graph. Due to inherent difference, techniques in [11, 21] are not applicable to the problem studied in the paper. Moreover, they do not exclude disconnected structure matchings.

As mentioned earlier, grafil [24] (the benchmark technique in our experimental evaluations) directly targets the subgraph search. Very recently, [25] proposes a general framework for conducting edit-distance based filtering in graph structure search; nevertheless, they also showed the pruning power is weaker than Grafil when applying to subgraph similarity search with the distance threshold σ is not greater than 3. Both Grafil and the work of [25] do not support effectively for *mccs*-based similarity search (i.e. connected subgraph search) and do not exclude disconnected substructure matchings.

[13, 20] are the only existing techniques that enforce connected substructure similarity search. gString techniques are used in [13] to provide approximate solutions to the problem studied in the paper. Nevertheless, gString based

techniques may miss the correct answers; thus they are not applicable to our problem that targets exact solutions for connected substructure similarity search. Most recently, [20] investigates the reverse version of the problem in the paper. Observe that for a query graph to nearly contain a data graph, it must contain a larger portion of a data graph. [20] proposes to enumerate all neighborhood graphs of each data graphs and then index them together by conducting prefix sharing. Then, conduct super-graph containment search from a query graph to those shared prefixes. The techniques in [20] are not applicable to our problem in the paper since the computation logic is inherently different. Moreover, techniques in [20] do not support filtering.

The connected common subgraph detection problem is mainly conducted for maximum *induced* subgraph only. Related work may be found in [1, 2, 3, 4, 7, 8, 16, 14, 17]. These techniques are not applicable to MCCS detection.

8. 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.

Acknowledgment

The work was supported by ARC Grants DP0987557 and DP0881035 and Google Research Award. The third author was supported by ARC Discovery Grants DP0987273 and DP0881779. The fourth author was supported by RGC Hong Kong, No. 419008.

9. REFERENCES

[1] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068, 1986.

[2] I. M. Bomze, M. Budinich, M. Pelillo, and C. Rossi. Annealed replication: a new heuristic for the maximum clique problem. *Discrete Applied Mathematics*, 121(1-3):27–49, 2002.

[3] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[4] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *SSPR/SPR*, pages 123–132, 2002.

[5] 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.

[6] 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.

[7] D. Conte, C. Guidobaldi, and C. Sansone. A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In *GbRPR*, pages 130–141, 2003.

[8] P. J. Durand, R. Pasari, J. W. Baker, and C. che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2, 1999.

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

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

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

[12] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.

[13] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. Gstring: A novel approach for efficient search in graph databases. In *ICDE*, pages 566–575, 2007.

[14] E. B. Krissinel and K. Henrick. Common subgraph isomorphism detection by backtracking search. *Softw. Pract. Exper.*, 34(6):591–607, 2004.

[15] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

[16] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341–352, September 1972.

[17] J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Softw., Pract. Exper.*, 12(1):23–34, 1982.

[18] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *SIGKDD*, pages 647–652, 2004.

[19] 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.

[20] H. Shang, K. Zhu, X. Lin, Y. Zhang, and R. Ichise. Similarity search on supergraph containment. In *ICDE*, pages 637–648, 2010.

[21] D. W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *ICDE*, pages 976–985, 2007.

[22] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

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

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

[25] 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.

[26] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *ICDE*, pages 966–975, 2007.

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

[28] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *EDBT*, pages 181–192, 2008.