Similarity Search on Supergraph Containment

Haichuan Shang #1, Ke Zhu #2, Xuemin Lin #3, Ying Zhang #4,Ryutaro Ichise *5

#*University of New South Wales, Australia*
[1]shangh@cse.unsw.edu.au
[2]kez@cse.unsw.edu.au
[3]lxue@cse.unsw.edu.au
[4]yingz@cse.unsw.edu.au

**National Institute of Informatics, Japan*
[2]ichise@nii.ac.jp

***Abstract*— A supergraph containment search is to retrieve the data graphs contained by a query graph. In this paper, we study the problem of efficiently retrieving all data graphs approximately contained by a query graph, namely similarity search on supergraph containment. We propose a novel and efficient index to boost the efficiency of query processing. We have studied the query processing cost and propose two index construction strategies aimed at optimizing the performance of different types of data graphs: top-down strategy and bottom-up strategy. Moreover, a novel indexing technique is proposed by effectively merging the indexes of individual data graphs; this not only reduces the index size but also further reduces the query processing time. We conduct extensive experiments on real data sets to demonstrate the efficiency and the effectiveness of our techniques.**

I. Introduction

Graphs can be used to model complicatedly structured data from a wide range of applications such as Bioinformatics, Pattern Recognition, XML, Communication Network, Chemistry, Social Network, World Wide Web, etc. Many techniques have been developed to accommodate the demand for effectively managing and analyzing graph data. Graph containment search is important and fundamental to to identify the relationships among graphs. It consists of the following two problems:

1) Subgraph containment search: Given a graph database $D = \{g_1, g_2, ..., g_n\}$ and a query graph q, retrieve all graph $g_i \in D$ such that q is a subgraph of g_i.
2) Supergraph containment search: Given a graph database $D = \{g_1, g_2, ..., g_n\}$ and a query graph q, retrieve all graph $g_i \in D$ such that q is a supergraph of g_i.

Driven by many applications, considerable efforts have been witnessed from database research community to tame the intrinsic complexity of these two problems - both are NP-complete [1]. Many indexing and query processing techniques have been proposed to solve these two problems [2], [3], [4], [5], [6], [7], [8], [9].

In real applications, it may often be desirable to approximately conduct a graph contain search (i.e. similarity search) due to the following reasons. Firstly, graph data may not be error-free. Consequently, it is desirable to provide a set of result candidates for graph containment search. Secondly, the nature of some applications require similarity search. For example, in computer vision[10], [8], it is very common to model real objects into spatial parts and then connect them as edges based on the relationships between the parts. Given a databases of these objects represented by graphs, we can model a photo or other real world scene into a query graph and pass this query into the database to search what objects are contained by the query. In such applications, most time the objects stored in the database and query hardly have an exact containment relationship; an approximate search (or similarity search) is more desirable.

Approximate subgraph containment search has been investigated recently. Two techniques, Grafil and C-Tree, are proposed [11], [3]. In [3], an efficient C-Tree technique is proposed to conduct the approximate subgraph containment search based on edit distances; nevertheless, there is no guarantee to provide a precise answer to approximate subgraph containment search. Grafil [11] effectively develops a feature based pruning technique to conduct the approximate subgraph containment search based on the number of missing edges. The key in Grafil is to mine discriminative features in database graphs. Due to the exclusive pruning logic [8] used in supergraph containment search, it is infeasible to identify which features are discriminative in the approximate supergraph containment search regarding missing edges; thus, it is not applicable to the approximate supergraph containment search regarding missing edges.

Motivated by the facts that the approximate supergraph containment search regarding missing edges is as important, if not more, as the approximate subgraph containment search while the existing techniques can not be effectively applied, we study the problem of approximate supergraph containment search in this paper. To the best of our knowledge, we are the first to study this problem. We propose a novel index, global SG-Enum index, which clusters the subgraphs of the data graph into a tree of subgraphs. We also analyze the complexity of our query processing technique when using the technique, and propose efficient algorithms to improve the efficiency of the index in terms of storage space and query response time. Our work is based on the following observation.

If a database graph g is approximately contained by a query graph q, q must contain a subgraph of g, say, g' such that the difference between g and g' must be less than the user specified error tolerance threshold, σ. We can conclude g is

approximately contained by q if we are able to find at least one such g'. However, this problem is NP-complete. Looking for such subgraphs while processing queries can cause very slow query response. By using some indexing techniques, we can compute these subgraphs of database graphs in the preprocessing stage to speed up the query response time. In addition, subgraphs of the same graph may share a large portion of data. By using an index structure which can utilizes this sharing, we can reduce the index size and we can even share query computation. Furthermore, we observe that using different strategies to optimize the index will significantly affect the performance on different types of datasets. Our main contributions are summarized as follows:

1) We propose to convert the underlying problem, maximum common subgraph detection problem, into a σ-missing subgraph detection problem. We build a novel index structure, named SG-Enum index, to speed up the query response time.
2) We propose a novel algorithm SigmaCSDetection to detect σ-missing subgraphs, based on SG-Enum index.
3) We analyze the computational cost of the SigmaCSDetection. Two optimization strategies are proposed based on the cost model, namely, top-down and bottom-up strategies.
4) Since the graph database contains a large number of graphs, we propose efficient algorithms to merge the local SG-Enum index into a global SG-Enum index. The global index not only saves the space but also reduces the query processing cost.

We conducted comprehensive experiments on the real datasets as well as synthetic datasets to show the efficiency, effectiveness and scalability of our technique. We demonstrate that our technique can be up to two orders of magnitude faster than directly detecting σ-missing subgraphs in online computation. In addition, we show that the top-down algorithm is suitable for applications where the labels are uniformly distributed, whereas the bottom-up algorithm is designed for the applications where the label distribution is biased.

The rest of the paper is organized as follows. Section II presents the problem statements. Section III introduces the framework of our technique. Section IV proposes SigmaCS-Detection algorithm and SG-Enum index. Section V discusses how to merge local indices into a global index. Section VI reports the experimental results. The related work and conclusion are given in Section VII and Section VIII.

II. Preliminaries

It is worth mentioning that all graphs in the paper are simple graphs, because most of the graphs in real applications have this property. A graph is *simple* if it has no self loops nor multiple edges.

A. Supergraph Containment Query

Given two sets of labels, Σ_V and Σ_E, a *labeled* graph g is defined as a triple (V, E, l) where V is the set of vertices, $E \subseteq V \times V$ is the set of undirected edges, and l is a labeling function: $V \rightarrow \Sigma_V$ and $E \rightarrow \Sigma_E$. We denote the vertex set and the edge set of a graph g as $V(g)$ and $E(g)$, respectively. Given an edge $(u,\ v) \in E(g)$ and the mapping function l of g, $l(u)$, $l(v)$ are the labels of u and v in g and $l(u, v)$ is the label of the edge (u, v) in g. We use $|V(g)|$ and $|E(g)|$ to represent the number of vertices and the number of edges, respectively. In this paper, *subgraph* always means connected subgraph.

Definition 1: (SUBGRAPH ISOMORPHISM) Given two graphs $g' = (V', E', l')$ and $g = (V,\ E,\ l)$, g' is *subgraph-isomorphic* to g, denoted as $g' \subseteq g$, if there is an injective function $f:\ g' \rightarrow g$ such that

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

A graph g' is a *subgraph* of g if g' is subgraph-isomorphic to g where g is also called a supergraph of g', denoted by $g' \subseteq g$. We may also simply say that g contains g'.

Definition 2: (SUPERGRAPH CONTAINMENT QUERY) Given a graph database $D = \{g_1, g_2, ..., g_n\}$ and a query graph q, the problem of supergraph containment query is to find a set of graphs which are subgraph-isomorphic to q from D, such as $D_q = \{g | g \in D \wedge g \subseteq q\}$.

B. Graph Similarity

Definition 3: (Maximum Common Subgraph - MCS) Given two graphs g_1 and g_2, the maximum common subgraph of g_1 and g_2 is the largest *connected* subgraph of g_1 that is subgraph-isomorphic to g_2, denoted as $g' = mcs(g_1, g_2)$.

Note that in Definition 3, the size of a graph is measured by the number of edges.[1] Subgraph similarity is measured by the difference between a data graph g and MCS (q, g) where g is a data graph, called *subgraph distance*.

Definition 4: (Subgraph Distance) Given a query graph q and a data graph g, the Subgraph Distance is defined as,

$$dis(q,\ g)\ =\ |g|\ -\ |mcs(q,\ g)|.$$

Here, $|g|$ and $|mcs(q, g)|$ denote the number of edges in g and $mcs(q, g)$, respectively.

Note that $mcs(q, g)$ has the reflectivity, that is $mcs(q, g) = mcs(g, q)$. Nevertheless, this reflectivity does not hold for $dis(q, g)$, that is, $dis(q,\ g) \neq dis(g, q)$ unless $|q| = |g|$.

Definition 5: (Graph Similarity) Given a query graph q and a data graph g, the similarity is defined by,

$$sim(q,\ g)\ =\ 1 - \frac{dis(q,\ g)}{|g|}$$

.

Note that $sim(q,\ g) \in [0, 1]$ because $|g|\ \geq\ |mcs(q,\ g)|$. The larger $sim(q,\ g)$, the similar the two graphs. As graph similarity can be simply converted to subgraph distance by $dis(q,\ g)\ =\ (1 - sim(q,\ g))\ \times\ |g|$, the techniques on computing subgraph distance also can be immediately applied to computing graph similarities.

[1] In MCS classification[12], our definition is classified to the connected MCES category.

C. Problem Statement

Definition 6: (Supergraph Similarity Search) Given a graph database $D = \{g_1,\ g_2, ...,\ g_n\}$, a query graph q, and a subgraph distance threshold σ, the subgraph similarity search is to retrieve all the graphs $g_i \in D$ with $dis(q,\ g_i) \leq \sigma$.

For representation simplicity, graph refer to undirect vertex-labeled graph in the rest of the paper; nevertheless, all the techniques can be immediately extended to cover directed and/or edge-labeled graphs.

III. Framework

To solve supergraph similarity search, our algorithm consists of the following phases.

1) We convert the underlying problem of supergraph similarity search, maximum common graph problem into a σ-missing subgraph detection problem, where σ is the error tolerance threshold. It means at most σ edges can be missed from the query. For each database graph g, we enumerate its σ-missing subgraphs. We use a tree structure to represent the σ-missing subgraphs for each g. We call it local SG-Enum index. The construction strategies of SG-Enum index are optimized based on the cost analysis of our local detection algorithm, SigmaCSDetection.
2) Having constructed local SG-Enum indices for all database graphs, we merge them into a global SG-Enum index. The global SG-Enum index is a tree based structure. Each node in the global index could be shared by many local indices. Each node is also attached with an ID list to record which local indices share this node.
3) We use the global version of SigmaCSDetection to effectively answer supergraph similarity search. The global SigmaCSDetection utilizes the global SG-Enum index to share computational cost between all data graphs.

IV. Sigma-Missing Common Subgraph Detection

In this section, we will firstly convert the underlying problem, maximum common subgraph detection, into a σ-missing subgraph detection problem. Secondly we will introduce a straightforward algorithm to test the existence of σ-missing subgraphs. Thirdly, we will introduce the local SG-Enum index to index σ-missing subgraphs and SigmaCSDetection algorithm to process queries based on local SG-Enum index. At last, we analyze the cost of SigmaCSDetection and present two index construction strategies based on the cost model.

To answer supergraph similarity queries, efficiently finding the maximum common subgraph is the fundamental problem. The naive method is to find the MCS between q and every $g_i \in D$ one by one. The issue is that finding MCS is well-known to be a NP-complete problem. MCS is also not indexable since q is unknown.

In supergraph similarity search, we only need to know whether $|g| - |mcs(q,g)| \leq \sigma$ is satisfied. Therefore, instead of computing $|mcs(q,g)|$ and then verifying the inequity $|g| - |mcs(q,g)| \leq \sigma$, we will convert the problem into detecting whether there exists a common subgraph $cs(q,g)$ such that $|g| - |cs(q,g)| \leq \sigma$. According to the definition of MCS, $|mcs(q,g)| \geq |cs(q,g)|$ always holds, because $mcs(q,g)$ is the maximum common subgraph on q and g. Thus, $|g| - |mcs(q,g)| \leq \sigma$ is satisfied if there exists a $cs(q,g)$ such that $|g| - |cs(q,g)| \leq \sigma$.

Definition 7: (σ-Missing Common Subgraph Detection) For a given graph g, a query graph q, and a threshold σ, the **σ-Missing Common Subgraph Detection** is to detect if there exists a common subgraph $cs(q,g)$ such that $|g| - |cs(q,g)| \leq \sigma$.

First of all, we briefly introduce the straightforward algorithm sketch. The algorithm for σ-Missing Common Subgraph Detection is in a DFS-Style algorithnm[2] as shown in Algorithm 1. For every edge e in graph g, the algorithm starts with mapping it to all possible candidates in q. Then for each of these mappings, the algorithm will try to extend the mapping by adding more edges into it. This is a DFS enumeration process. At any stage of the search, if the algorithm finds a cs such that $|g| - |cs(q,g)| \leq \sigma$, the algorithm is terminated and returns true. Edge e will be removed from search space at the end of each iteration since no answer will include this edge.

Algorithm 1: **DirectSigmaCSDetection** (q, g, σ)

```
Input  : q is a query graph;
         g is a data graph;
         σ is the threshold;
1 for each edge e in g do
2     for each mapping of e in q do
3         cs := e;
4         Extend cs and its mapping in both g and q;
5         if |g| − |cs| ≤ σ then
6             return true
7     Remove e from g;
8 return false
```

The algorithm 1 has the following disadvantages when it is used for supergraph similarity search.

1) All the common subgraphs of q and g have to be enumerated. However, our study shows that we only need to enumerate a subset of these subgraphs.
2) The the common subgraph cs is extended randomly. Actually, the order to extend cs will drastically affect the detection cost. This has been shown in [5].
3) The data graphs $D = \{g_1, g_2, ..., g_n\}$ are tested one by one. According to our study, most the testing operations can be shared and the cost can be further reduced.

A. SG-Enum Index

We propose a novel tree structured index, named SG-Enum, which enumerates all the σ-missing subgraphs. In similarity search, σ is only meaningful when it is small. Therefore, we could construct one SG-Enum index for each σ value.

[2]The clique based algorithms are not designed for the MCES similarity measurements[13], [14], [15]

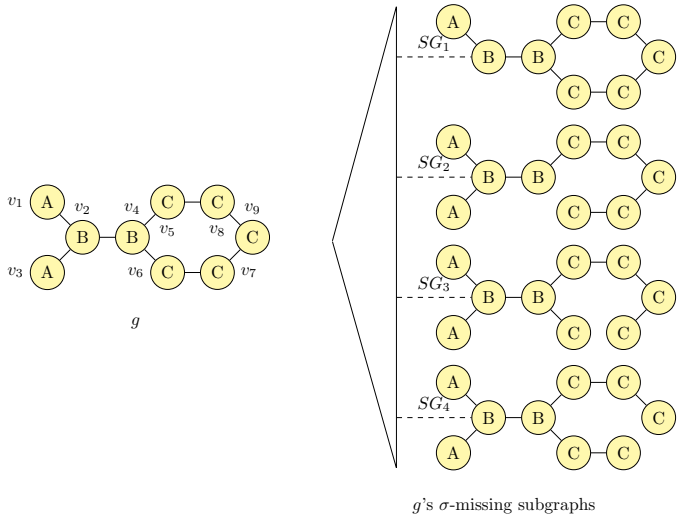


Fig. 1. A data graph g and its σ-missing subgraphs

An example of SG-Enum index is shown in figure 1. The graph g in the left side is a data graph. The right side of figure shows the framework of SG-Enum index. Assuming the threshold $\sigma = 1$, the data graph g has four subgraphs which miss exactly one edge. These subgraphs are called *g's σ-missing subgraphs*(Isomorphic subgraphs are removed). The four subgraphs are shown as SG_1, SG_2, SG_3 and SG_4. In σ-missing common subgraph detection, $|g| - |cs(q, g)| \leq \sigma$ is satisfied if and only if at least one of these four subgraphs is contained by q. When a query graph q arrives, the SG-Enum index serves to efficiently answer whether q contains g's σ-missing subgraphs.

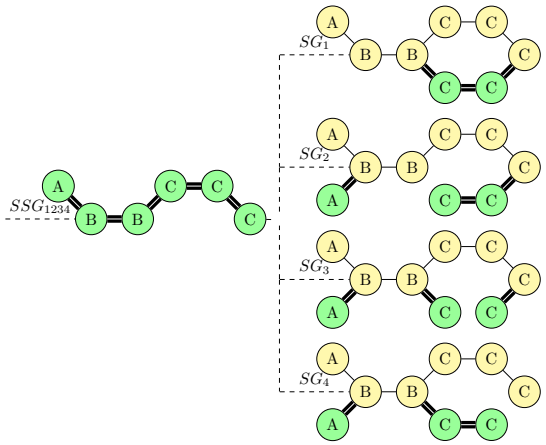


Fig. 2. An example of SG-Enum index

A sample of SG-Enum index of the data graph g is shown in figure 2. The SG-Enum index is in a tree structure. The leaf nodes are the σ-missing subgraphs of g. The σ-missing subgraphs are clustered to a tree of subgraphs. The non-leaf nodes are the maximum common subgraphs of its children, denoted as SSG_x. For example, the node SSG_{1234} is the maximum common subgraph of SG_1, SG_2, SG_3, SG_4. If such root node does not exist, we create a virtual node as the root. In the SG-Enum, each subgraph represented by a node has a set of edges and vertices which do not appear in its parent node. They are displayed as double lines and green vertices in the figure. We call them *extension edges* .

B. Query Processing and Index Efficiency

It is clear that, if a query graph does not contain the graph represented by a node, the query graph will not contain any of the graphs represented by its descendants. Another observation is that $|g| - |mcs(q, g)| \leq \sigma$ is satisfied if and only if q contains at least one graph represented by a leaf node in g's SG-Enum index. Based on these two observations, we propose our σ-missing common subgraph detection algorithm in algorithm 2 and algorithm 3. Firstly, we enumerate all the subgraph isomorphic mapping of the graph represented by the root node in line 1-2 of algorithm 2. If the root node is a σ-missing subgraph of g and it is an isomorphic subgraph of q, then we terminate the algorithm. Otherwise, we do a DFS search by calling SigmaCSDetectionNode for each child of the root node. In SigmaCSDetectionNode, we try to match the extension edges by extending its parent node's mapping in q(Line 1-2). For each successful matching, we recursively call SigmaCSDetectionNode to traverse the SG-Enum index(Line 5-7). The algorithm is terminated if one of the graph represented by a leaf node is contained by the query graph q(Line 3-4) or we know the graph g will not be part of the answer if the whole index has been traversed and no isomorphic σ-missing subgraph has been found for g.

Algorithm 2: **SigmaCSDetection** (q, I_g, σ)

```
Input  : q is a query graph;
         I_g is the SG-Enum index of g's σ-missing subgraphs;
         σ is the threshold;
1 SSG = I_g.root;
2 for each subgraph isomorphic mapping iso of SSG in q do
3     if SSG is a leaf node then
4         return true
5     for each child SSG' of SSG do
6         if SigmaCSDetectionNode (q,iso,SSG') then
7             return true
8 return false
```

(**Correctness**)It can be immediately verified that $|g| - |mcs(q, g)| \leq \sigma$ is satisfied *iff* there exist a common subgraph $cs(q, g)$ such that $|g| - |cs(q, g)| \leq \sigma$. It is equivalent to that there exist a σ-missing subgraph SG_i of g which are contained by q.

(**Cost Analysis**) The algorithm 2 follows depth-first search strategy. We define the *search breadth* of a node SSG in respect to a query graph q below, denoted by B_{SSG}.

Definition 8: (SEARCH BREADTH) Given a node SSG in the SG-Enum index I_g of a data graph g and a query graph q, the search breadth B_{SSG} from SSG to q is defined as $B_{SSG} = |\{f | f : SSG \rightarrow q\}|$ where f is the subgraph isomorphic mappings from SSG to q.

Algorithm 3: **SigmaCSDetectionNode** (q, iso, SSG)

```
Input : q is a query graph;
        iso is the mapping of the parent node to q;
        SSG is the current node;
Match the extension edges of ssg on the mapping iso in q;
for each matched mapping iso' do
    if SSG is a leaf node then
        return true
    for each child SSG' of SSG do
        if SigmaCSDetectionNode (q,iso',SSG') then
            return true
return false
```

Given a node SSG, one of its children SSG' and a query graph q, if we have already known the subgraph isomorphic mappings from SSG to q, we define the cost of matching SSG' to q as the *isomorphic mapping testing cost* of node SSG', denoted as $T_{iso}(SSG, SSG', q)$. Let E is the extension edges of SSG', $E = E(SSG') - E(SSG)$. Let E to an ordered set $\overrightarrow{E} = \{e_1, e_2, ..., e_n\}$ where $n = |E|$, then the isomorphic mapping testing cost of node SSG' can be calculated as follows.

$$T_{iso}(SSG, SSG', q) = deg_{avg} \cdot (B_{SSG} + B_{SSG\cup\{e_1\}} + B_{SSG\cup\{e_1,e_2\}} + ... + B_{SSG\cup(E-\{e_n\})})$$

where deg_{avg} is the average degree of the query graph.

It is important to note that $B_{SSG\cup\{e\}}$ is depending on B_{SSG}, since every subgraph isomorphic function $f_{SSG\cup\{e\}} : SSG \cup \{e\} \rightarrow q$ is extended from $f_{SSG} : SSG \rightarrow q$. Therefore, the order of $\overrightarrow{E}$ will significantly affect the cost.

The overall cost of σ-missing common subgraph detection between q and g using SG-Enum index is as follows.

$$T_{SigmaCS} = \sum_{SSG\in I_g} \sum_{SSG'\in SSG.child} T_{iso}(SSG, SSG', q) \quad (1)$$

Intuitively, it is not difficult to see that the cost of SigmaCSDetection is greatly dependent on two factors: i) the total number of $SSG \in I_g$. ii) the cost of individual $T_{iso}(SSG, SSG', q)$. However, finding the optimal SG-Enum index is NP-complete. Based on above intuition, we will introduce two heuristic construction strategies.

C. Index Construction

Although it is unrealistic to find the optimal SG-Enum index, we can still improve the query processing efficiency by optimizing the structure and edge enumeration order of the SG-Enum index. If we use B_{avg} to represent the average search breadth, equation 1 can be approximated as follows.

$$T_{SigmaCS} = M \cdot B_{avg}$$

where M is the number of extension edges in each node in the SG-Enum index.

There are two strategies to improve the index efficiency: reducing the number of extension edges or reducing the search breadth. Based on these two different strategies, we develop two index construction algorithms, the top-down algorithm and bottom-up algorithm.

1) Top-Down Algorithm: The top-down algorithm optimize the index by reducing the number of extension edges in the SG-Enum index. It aims to share the computational cost as much as possible.

Given a data graph g, I_g is the SG-Enum index of g. For each node $SSG \in I_g$, $ExtE(SSG)$ represents the number of extension edges in SSG.(Remember the double line shown in figure 2) We use M to represent the total number of extension edges in the index.

$$M = \sum_{SSG\in I_g} ExtE(SSG)$$

For the purpose of reducing the number of extension edges, the optimal SG-Enum index is the index of g with the minimal value of M. Note that the σ-missing subgraphs(i.e. the leaf nodes) are also considered as a tree node in the SG-Enum index. In the example shown in figure 2, the total number of extension edges is,

$$\begin{aligned} M =& ExtE(SSG_{1234}) + ExtE(SG_1) + \\ & ExtE(SG_2) + ExtE(SG_3) + ExtE(SG_4) \\ =& 5 + 3 + 3 + 3 + 3 \\ =& 17 \end{aligned}$$

Theorem 1: Finding the optimal SG-Enum index with minimal M is NP-hard.

Proof: In this proof, we will show that a special case of this problem is actually the weighted set cover problem. Suppose we want to find an optimal 3-level SG-Enum index for a group of σ-missing subgraphs and there does not exist a maximum common subgraph for the whole group. The root node will be an empty virtual node. The leaf level contains all σ-missing subgraphs $\mathcal{SG} = \{SG_i\}$ where $1 \leq i \leq n$ and n is the total number of σ-missing subgraphs. The level two nodes contain a set of maximum common subgraphs, $\mathcal{SSG} = \{SSG_x\}$ where x is the set of the IDs of its children. Formally, the set of level two nodes are $\mathcal{SSG} \subseteq 2^{\mathcal{SG}}$ and $\bigcup_{SSG_x\in\mathcal{SSG}} x = \{1, 2, ..., n\}$. The total cost M is,

$$\begin{aligned} M &= \sum_{SSG_x\in\mathcal{SSG}} ExtE(SSG_x) + \sum_{SG_x\in\mathcal{SG}} ExtE(SG_i) \\ &= \sum_{SSG_x\in\mathcal{SSG}} (ExtE(SSG_x) + \sum_{i\in x} ExtE(SG_i)) \end{aligned}$$

If we let $w(SSG) = ExtE(SSG_x) + \sum_{i\in x} ExtE(SG_i)$, it is a nonnegative function $w : SSG \rightarrow \mathbb{R}$. Then,

$$M_{\mathcal{SSG}} = \sum_{SSG\in\mathcal{SSG}} w(SSG)$$

Given a set of σ-missing subgraphs $\mathcal{SG} = \{SG_1, SG_2, ..., SG_n\}$, finding a cover set $\mathcal{SSG}$, such that $M_{\mathcal{SSG}}$ is minimal, is the weighted set cover problem.

■

We propose a heuristic algorithm in algorithm 4 and algorithm 5. Algorithm 4 is the main function. In line 1, it generates all g's σ-missing subgraphs and their maximum common graph. The MCS will be the root node of the global index tree. We will introduce a virtual node if no such MCS could be found. All of these subgraphs and the root node will be passed to algorithm 5 as an input. Algorithm 5 will divide these subgraphs into two groups to maximize the sharing of extension edges. In line 1, it tries to find an edge which will be shared by the maximum number of subgraphs. If this edge is only shared by one subgraph, that means there are no common edges shared between any pair of the subgraphs. Therefore, we just need to add all of them to the parent node and terminate the algorithm. Otherwise, we will extend the graph represented by the parent node with the selected edge. This extended graph will become the new children node. The subgroups are then divided into two groups. One group are those with the shared edge and another are those without. These two groups are then subsequently further divided by recursively calling algorithm 5 if they contain more than one subgraph.

Algorithm 4: **TopDownAlgorithm** (g,σ)

Input : q is a query graph;
σ is the threshold;
Output : I_g is the SG-Enum index of g's σ-missing subgraphs;

1 $\mathcal{SG} = g$'s σ-missing subgraphs;
2 $I_g.SSG_{root}$ =the maximum common subgraph of all SGs in $\mathcal{SG}$;
3 TopDownMCS ($\mathcal{SG}$, SSG_{root});
4 **return** I_g

Algorithm 5: **TopDownMCS** ($\mathcal{SG}$, SSG)

Input : $\mathcal{SG}$ is a set of graphs;
SSG is a node in the SG-Enum index;
Output : I_g is the SG-Enum index of g's σ-missing subgraphs;

1 e is the edge maximize the value $m = |\{SG|SG \in \mathcal{SG} \wedge SSG \cup \{e\} \subseteq SG\}|$;
2 **if** $m == 1$ **then**
3 Add all SGs in $\mathcal{SG}$ as SSG's children;
4 **return**
5 **else**
6 Add $SSG' = SSG \cup \{e\}$ as SSG's child;
7 $\mathcal{SG}' = \{SG|SG \in \mathcal{SG} \wedge SSG \cup \{e\} \subseteq SG\}$;
8 $\overline{\mathcal{SG}'} = \mathcal{SG} - \mathcal{SG}'$;
9 TopDownMCS ($\mathcal{SG}'$, SSG');
10 TopDownMCS ($\overline{\mathcal{SG}'}$, SSG);
11 **return**

For example, we will construct a SG-Enum index for the graph g in figure 1 by using the top-down strategy. In the first step, we set the maximum common subgraph of all g's σ-missing subgraphs, SG_{1234}, as the root node. Among the remained edges, (v_2, v_3), (v_4, v_6), (v_6, v_7), (v_7, v_9) are all shared by 3 subgraphs, however (v_6, v_7) is not connected to SG_{1234}. Therefore we will randomly choose one from the three remaining candidates. In this example, we choose (v_2, v_3). The σ-missing subgraphs are now divided into two subgroups $\mathcal{SG}' = \{SG_2, SG_3, SG_4\}$ and $\overline{\mathcal{SG}'} = \{SG_1\}$. As SG' has more than one member, the algorithm will try to divide it again until there are no subgroups with more than one subgraph. The resulted local SG-Enum index is shown in figure 3 (Due to space limit, we omit the vertex IDs in this figure. Please refer to figure 1 for vertex IDs).

In this example, the total number of extension edges is,

$$\begin{aligned} M =& ExtE(SSG_{1234}) + ExtE(SSG_{234}) + \\ & ExtE(SSG_{34}) + ExtE(SG_1) + ExtE(SG_2) + \\ & ExtE(SG_3) + ExtE(SG_4) \\ =& 5 + 1 + 1 + 3 + 2 + 1 + 1 \\ =& 14 \end{aligned}$$

(Cost Analysis) The top-down algorithm consists of two steps, finding the maximum common subgraph and finding the most commonly shared edge for every level in the index. In the first step, the complexity of finding the set of common edges is $O(mn)$ and finding the maximum connected subgraph from the set is $O(n^2)$ where n is the number of edges and m is the number of σ-missing subgraphs. In the second step, we can find the most commonly shared edge e in $O(nm)$ time for each level. There are at most n levels. Therefore, the overall complexity is $O(mn + n^2 + mn^2) = O(mn^2)$.

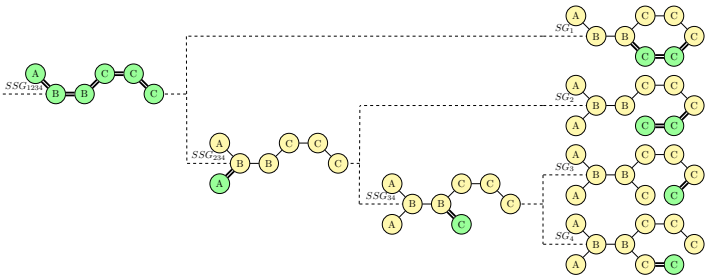


Fig. 3. SG-Enum index constructed by top-down algorithm

2) Bottom-Up Algorithm: The bottom-up algorithm optimize the index by reducing the search breadth. The edges in each σ-missing subgraph are sorted to minimize the search breadth. Due to each σ-missing subgraph is optimized for itself. The SG-Enum index constructed by this algorithm may contain more extension edges than the one constructed by the top-down algorithm. However, if the label distribution is biased, the benefit from reducing search breadth is more than the cost resulted from the increased number of extension edges.

Determining the best edge order for subgraph isomorphic mapping test has been demonstrated to be an NP-complete problem[5] and the heuristical algorithm for generating an effective edge order for a given graph g has been proposed in the same paper. We employ their algorithm to determine edge order for each σ-missing subgraphs of g. We briefly introduce the criterions for edge order determination. The frequency of

an edge, $Freq(e)$, is the number of its appearance in the whole graph database.

1) The ordered edge set and visited vertex set are initially empty. The first edge e is the edge with the lowest $Freq(e)$. We add e to the ordered edge set. We add the two ends of e to the visited vertex set.
2) If there are edges whose two ends are both in the visited vertex set, we immediately add them to the ordered edge set in non-descending order of their frequencies.
3) Among the edges which has one and only one end in the visited vertex set, we select the one with the lower frequency into the ordered edge set and add its ends to the visited vertex set. If there are ties, we choose the one with the highest vertex degree. If there are still ties, we randomly choose one.

After the ordered edge sets are constructed for each σ-missing subgraph, we merge them into a prefix tree. The resulted prefix tree is the SG-Enum index. The algorithm is shown in algorithm 6.

Algorithm 6: **BottomUpAlgorithm** (g,σ)

Input : q is a query graph;
σ is the threshold;
Output : I_g is the SG-Enum index of g's σ-missing subgraphs;

```
SG = g's σ-missing subgraphs;
for each σ-missing subgraph SG_i in SG do
    EX_i = SG_i's ordered edge set;
I_g = the prefix tree of all EX_i;
return I_g
```

For demonstration purpose, we assume the frequencies of the edges for g in figure 1 are as follows.

$$Freq(A{-}B) < Freq(B{-}B) < Freq(B{-}C) < Freq(C{-}C)$$

The ordered edge set of the σ-missing subgraph SG_2 in figure 1 can be computed by the following steps. We select (v_1, v_2) as first edge because its label, $A-B$, has the lowest frequency. The visited vertex set is $\{v_1, v_2\}$ now. The edges with one end in the visited vertex set are (v_2, v_3) and (v_2, v_4). As the $Freq(A-B) < Freq(B-B)$, the next edge will be (v_2, v_3). Now the visited vertex set is $\{v_1, v_2, v_3\}$. After that, we select (v_2, v_4) since it is the only edge connected to the visited vertex set. Subsequently, the (v_4, v_5), (v_5, v_8), (v_8, v_9), (v_9, v_7), and (v_6, v_7) are chosen in order. The resulted ordered edge sets for SG_2 and other the σ-missing subgraphs are shown in figure 4. The numbers attached to edges are the their orders in the ordered edges sets. Having obtained all the ordered edge sets, we will merge them into a prefix tree. The SG-Enum index is shown in figure 5.

(Cost Analysis) In the bottom-up algorithm we compute the ordered edge sets and merge the sets into a prefix tree. Computing the ordered edge sets requires $O(mnd)$ time and inserting them into the index costs (mnc) time where n is the number of edges, m is the number of σ-missing subgraphs, d is the average number of edges which has one and only one end in the visited vertex set and c is the average number of children of the nodes in the index. Thus, the overall complexity is $O(mnd + mnc)$.

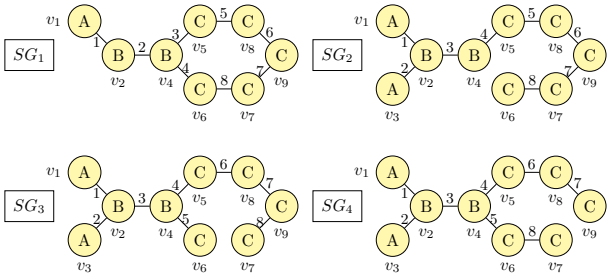


Fig. 4. Order Edge Set of σ-missing subgraphs

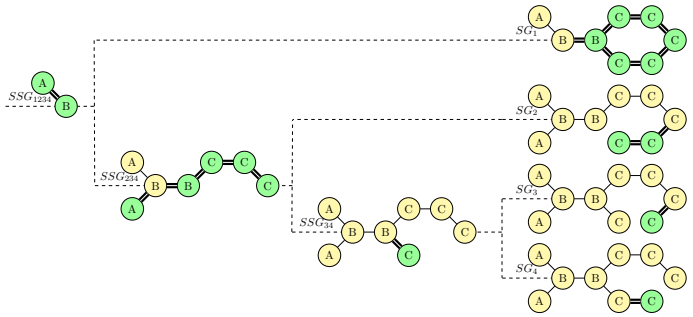


Fig. 5. SG-Enum index constructed by bottom-up algorithm

D. Top-Down versus Bottom-Up

The indices constructed by top-down and bottom-up strategies are expected to perform differently depending on whether the label distribution of the database is biased or not.

We say the label distribution is *biased* if the most database graphs contains a few edges whose frequencies are significantly lower than others in the same graph. The bottom-up strategy is more effective when label distribution of the database graphs is *biased*. For this type of datasets, the bottom-up strategy can significant reduce the search breadth, because for most database graphs, there exist a few edges which are significantly more selective than other edges in the same graph. If the label distribution is uniform, which means the frequencies of most edges are similar,(e.g. most edges are frequent or most edges are infrequent) the search breadth will not differ greatly no matter how the edges are ordered, because edges of many database graphs are similarly selective. Therefore, in this case, the bottom-up strategy will not as effective as the top-down strategy. For example, in the AIDS antiviral dataset, 5 edge labels represent 94% of all edge labels. This means many graphs in the dataset may only contain edges of those 5 labels, which are all very frequent. The top-down strategy will be more effective in this case.

V. Global SG-Enum index

We have discussed the σ-missing common subgraph detection algorithm between a query graph and a data graph.

In the graph database, we have a set of data graphs $D = \{g_1, g_2, ...g_n\}$. Instead of pairwisely detecting the σ-missing common subgraph between query graph q and each $g_i \in D$, We can combine the indices of these data graphs and process the query on the combined index for all data graphs.

A. Index Combination

Assuming we have a set of data graphs $D = \{g_1, g_2, ...g_n\}$, each data graph g_i has an independent SG-Enum index I_{g_i}. Now, we discuss how to merge all I_{g_i} for $1 \leq i \leq n$ into a global index I_D.

Before discussing the merging algorithm, we first describe the physical structure of SG-Enum index. In the previous examples, each node in the SG-Enum index may contain several extension edges. In physical structure, each node in the SG-Enum index only contain one extension edge. If a node contains n extension edges, we expand it to an n node path. For example, in figure 6, the node SSG_{12} can be expanded to SSG_{12A} and SSG_{12B}. Actually, instead of storing the whole subgraph in each node, we only store the extension edge. Therefore the space requirement is $O(\sum_{SSG \in I_g} ExtE(SSG))$.

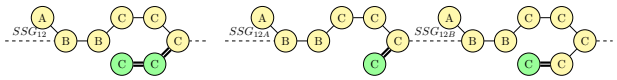


Fig. 6. The physical structure of SG-Enum index

Since each node is an edge in the physical structure of SG-Enum index, it is straightforward to merge the indices. We present our algorithm in algorithm 7 and algorithm 8. In algorithm 7 line 1, we create a $VirtualNode$ for the global SG-Enum index I_D. For each node in the global SG-Enum index, we use a sorted ID list to record the graphs whose local SG-Enum indices are stored as descendants of this node. In line 2, we set the ID list of the virtual node to be the whole graph database. In line 3-5, we call algorithm 8 to merge each local SG-Enum index I_{g_i} into I_D. In algorithm 8 line 1-6, we compare the extension edge of the current node SSG_g to the extension edges of SSG_D's children. If they are the same, it means there is an identical node already existing in I_D. In this case, we can insert the graph ID into $SSG_D.Child$'s ID list. Then we continue to merge all of SSG_g's children. If none of SSG_D's children has a same extension edge as SSG_g's, we add SSG_g as SSG_D's child and continue the merging process for all SSG_g's children. The newly added child's ID list is initialized to i, the graph ID.

Since each node only contains one extension edge in the physical structure of SG-Enum index, the comparing cost between two nodes is $O(1)$. We traverse each node in SG-Enum index I_{g_i} once, and comparing it to the children of the corresponding node in I_D. Therefore, the time complexity of the index combination algorithm is

$$O(c \cdot \sum_{I_{g_i} \in I} |I_{g_i}|)$$

where c is the average number of children per node in I_D and $|I_{g_i}|$ is the number of nodes in each SG-Enum index I_{g_i}.

Algorithm 7: **Combine** (I)

```
Input  : I = {I_g1, I_g2, ..., I_gn} is a set of SG-Enum index;
Output : I_D is the global SG-Enum index;
1  I_D.root = VirtualNode;
2  I_D.root.IDs = {i | g_i ∈ D};
3  for 1 ≤ i ≤ n do
4      CombineNode (I_D.root, I_gi.root, i);
5  return I_D
```

Algorithm 8: **CombineNode** (SSG_D,SSG_g, i)

```
Input  : SSG_D is a node in the global SG-Enum index;
         SSG_g is a node in the SG-Enum index;
1  for each child SSG_D.child of SSG_D do
2      if SSG_D.child = SSG_g then
3          SSG_D.child.IDs.insert(i);
4          for each child SSG_g.child of SSG_g do
5              CombineNode (SSG_D.child, SSG_g.child, i);
6          return
7  add SSG_g as a child SSG_D.child of SSG_D;
8  SSG_D.child.IDs = {i};
9  for each child SSG_g.child of SSG_g do
10     CombineNode (SSG_D.child, SSG_g.child, i);
11 return
```

An example is shown in figure 7 and figure 8. In figure 7, the three SG-Enum index I_{g_1}, I_{g_2} and I_{g_3} are corresponding to the local SG-Enum indices for g_1, g_2, g_3, respectively. We can merge them to the global SG-Enum index I_D shown in figure 8. The resulted ID lists are also shown next to each node in I_D.

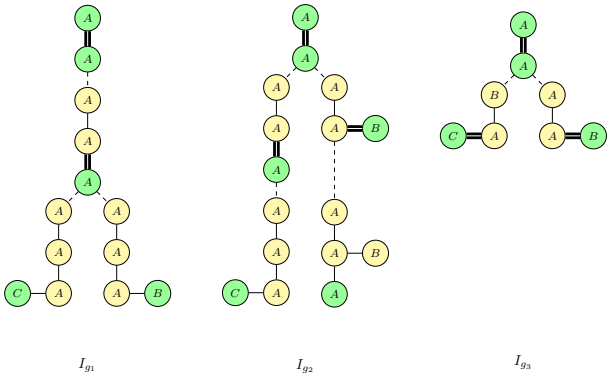


Fig. 7. A set of SG-Enum index

B. Query Processing

The query processing algorithm is shown in algorithm 9. Initially, SSG is the root node of I_D, and $match$ is a list to store the results and is initialized to be empty.. In line 1-2, we can skip all the descendants of the current node if all subgraphs

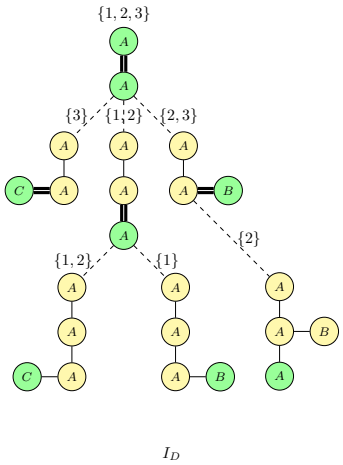


Fig. 8. The global SG-Enum index

contained in the current node are already included in the result. Otherwise, we need to check whether the graph represented by the current node is contained by the query graph. If so, in line 4-6, we will identify all the database graphs indexed by SSG's direct children. If SSG is a leaf node in any local SG-Enum index I_{g_i}, g_i must be indexed by SSG's ID list but not by any of SSG's children's ID lists. Line 7 recognizes all the leaf nodes and insert the corresponding graph IDs into the result. Finally, we recursively call the algorithm itself for all the children of the current node in line 8-9.

Algorithm 9: **QueryProcessing** (SSG,q,$match$)

Input : SSG is a node in the global SG-Enum index, initialized by $I_D.root$;
q is a query graph;
$match$ is the result list ;

1 **if** $SSG.IDs - match = \emptyset$ **then**
2 **return**
3 **if** SSG is sub-iso to q **then**
4 $R = \emptyset$;
5 **for** each child $SSG.child$ of SSG **do**
6 $R.insert(SSG.child.IDs)$;
7 $match.insert(SSG.IDs - R)$;
8 **for** each child $SSG.child$ of SSG **do**
9 QueryProcessing ($SSG.child$,q,$match$);
10 **return**

VI. Experiments

In this section, we performed extensive experimental study to demonstrate the effectiveness and efficiency of our proposed techniques. We tested the performance of global SG-Enum indices constructed by two optimization strategies, namely, **Top-down** and **Bottom-up**. Both types are using the query processing techniques described in Section V. We compared our techniques against DirectSigmaCSDetection algorithm.

Our experiments are conducted on the real datasets as well as synthetic datasets. Following previous works [5], [2], [3], [6], [4], we omit edge labels of graphs in our experiments.

Real dataset. We use the AIDS Antiviral Screen dataset, which consists of $43,905$ classified chemical molecules. The dataset is publicly available on the website of Development Therapeutics Program.

Synthetic dataset In order to evaluate how label distribution affects the two optimization strategies, we relabeled a subset of the vertices with infrequent labels.

A. Performance on Real Dataset

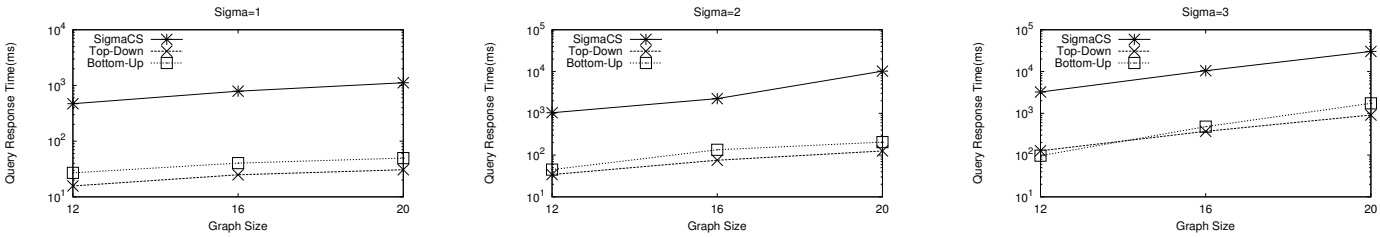
We examine the performance of SG-Enum index and SigmaCSDetection over the AIDS antiviral database — a popular benchmark in graph [7], [5]. There are totally 62 distinct vertex labels in the data set and top 5 labeled edges are shown in Figure 9. We extract three datasets of substructures from the AIDS antiviral database with 12, 16 and 20 edges in average, denoted as D12, D16 and D20, respectively. Each dataset contain 10K substructures. In scalability test, we randomly sample 1k, 2k, 5K, 10K, and 20K data graphs with 16 edges in average. The query set contains 1000 randomly chosen graphs from the AIDS antiviral database with 25 vertices and 27 edges in average.

Rank	Edge	Freq.	Rank	Edge	Freq.
1	C – C	57.2%	4	N – N	6.0%
2	C – N	15.2%	5	C – S	2.1%
3	C – O	13.4%			

Fig. 9. Edge Frequency of AIDS

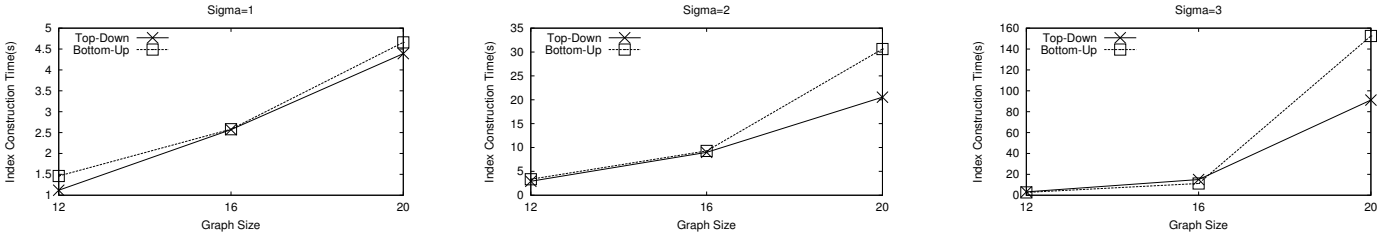
In the first experiment, we examine the efficiency of our techniques. Experiment results are shown in Figure 10. It demonstrates that both Top-down and Bottom-up algorithms outperform the DirectSigmaCSDetection algorithm by 1-2 orders of magnitude. the Top-down strategy performs better than the bottom-up strategy in nearly all settings. This is because, as shown in fig 9, the top 5 labels represent 94% frequencies. At first glance, this distribution seems to be very biased. In fact, this distribution is very similar to a uniform distribution containing only 5 labels with an insignificant portion of outliers. In other words, a major portion of the σ-missing subgraphs only contains these 5 labels. The bottom-up optimization is ineffective in this scenario because that all the edges are frequent in these σ-missing subgraphs. Therefore, the search breadth cannot be reduced drastically no matter how the edges are ordered. In the contrast, if there are lots of edges with frequent labels, there are more sharings among the σ-missing subgraphs. Therefore, the Top-down strategy are more effective in this case. In the synthetic experiment conducted below, we will show how the percentage of infrequent labels affects the performance.

We evaluate the index construction cost and index size for both Top-down and Bottom-up strategies. The results are shown in figure 11 and figure 12. We found both index construction strategies are very efficient in index construction. The Top-down strategy has faster construction time and smaller index size in most settings. As expected, the construction cost of both strategies increase significantly with σ or graph size



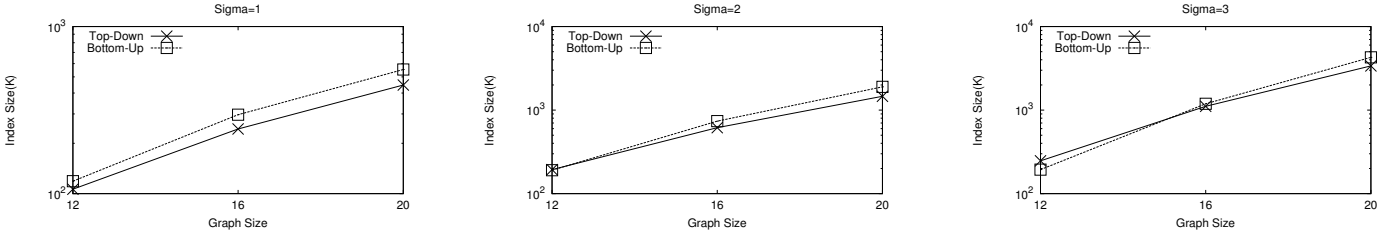
(a) Sigma = 1 (b) Sigma = 2 (c) Sigma = 3

Fig. 10. Query Response Time



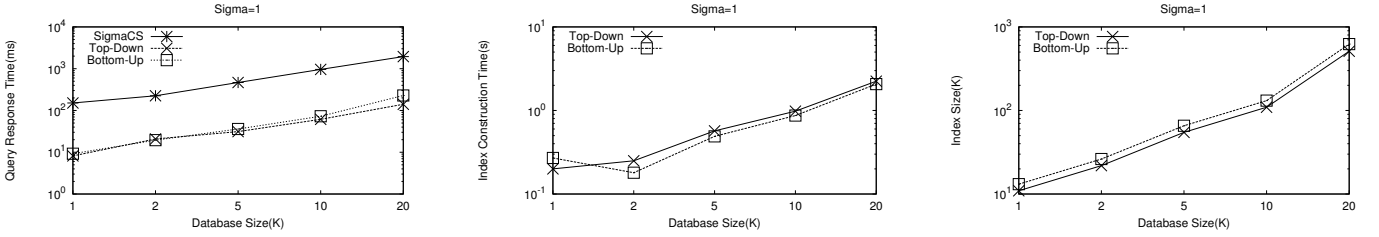
(a) Sigma = 1 (b) Sigma = 2 (c) Sigma = 3

Fig. 11. Index Construction Time



(a) Sigma = 1 (b) Sigma = 2 (c) Sigma = 3

Fig. 12. Index Size

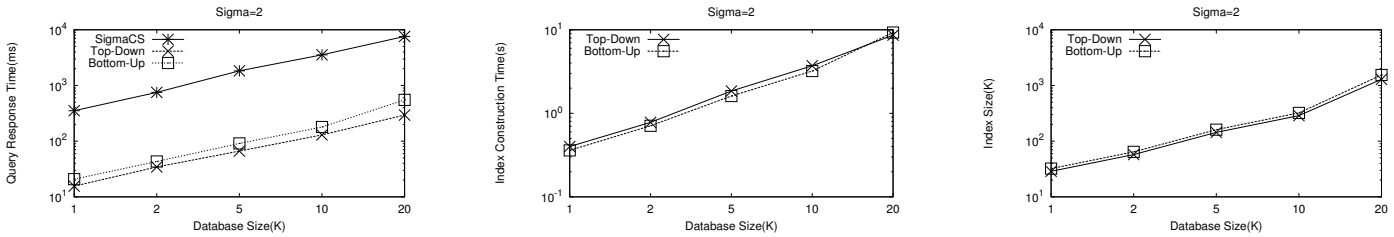


(a) Query Response Time (b) Index Construction Time (c) Index Size

Fig. 13. Scalability with $\sigma = 1$

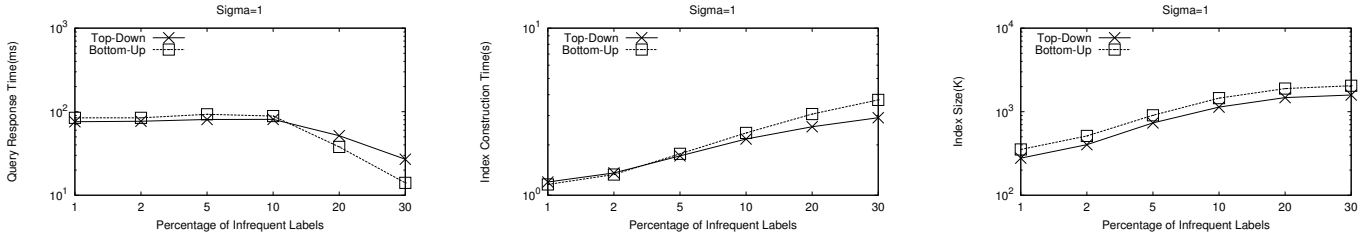
increase. It is because the number of σ-missing subgraphs is approximately equal to $\binom{n}{\sigma}$ where n is the number of edges in the graph. An interesting observation is that the top-down strategy outperforms the bottom-up strategy in both construction time and index size. This is because the top-5 labels are representing 94% of all labels in this dataset. As a result, the top-down strategy could find significant sharing at top few levels. After these shared parts have been removed from the σ-missing subgraphs, the remaining parts are mostly outliers and they share very little. Consequently, the top-down SG-Enum index is relatively effective in this case. However, the bottom-up strategy aims at optimizing the edge enumeration order for individual σ-missing subgraphs, therefore, it cannot enjoy the significant sharing as the top-down strategy can.

We evaluate the scalability against varying database sizes with fixed $\sigma = 1$ and $\sigma = 2$. We present the query response time, the index construction time and the index size in figure 13 and figure 14. All three algorithms are scalable in query response time. Both Top-down and Bottom-up strategies outperform the DirectSigmaCSDetection algorithm by one order of magnitude for $\sigma = 1$ and 1.5 orders of magnitude



(a) Query Response Time (b) Index Construction Time (c) Index Size

Fig. 14. Scalability with $\sigma = 2$



(a) Query Response Time (b) Index Construction Time (c) Index Size

Fig. 15. Number of Infrequent Labels with $\sigma = 1$

for $\sigma = 2$. The Top-down and Bottom-up algorithms achieve nearly same performance for $\sigma = 1$ whereas the Top-down strategy is a slightly faster than the Bottom-up strategy for $\sigma = 2$. Both Top-down and Bottom-up strategies are scalable in terms of index construction cost and index size. There are no major difference found in the scalability test.

B. Performance on Synthetic Dataset

We evaluate how the label distribution affects the two strategies in figure 15. We generate six synthetic datasets by relabeling the vertices in D16 with, 1%, 2%, 5%, 10%, 20% and 30%, percentage of infrequent labels respectively. The result shows that the top-down strategy is a slightly faster than the bottom-up strategy when the percentage of infrequent label is less 10%. As we increase the percentage of infrequent label from 10% to 30%, the bottom-up strategy outperforms the top-down strategy and the difference increases as the percentage increases. The bottom-up strategy is two times faster than the top-down strategy when 30% of the vertices have infrequent labels. The result supports our hypothesis that the bottom-up strategy performs better when label distribution is biased. The bottom-up strategy chooses to enumerate the most selective edges to reduce the search space drastically, thus, the processing cost is reduced drastically as well. Since the data graphs have 16 edges on average and $\sigma = 1$, there is a high probability that each σ-missing subgraph has at least one edge with infrequent label when then percentage of infrequent labels is high enough($> 10\%$) . When the number of infrequent labels is less than 10%., a major portion of σ-missing subgraph contain oly frequent labels and the bottom-up is ineffective as shown in the experiment on the real dataset. As expected, the top-down strategy also performs better performance in terms of index construction time and index size.

Summary. Our experiment demonstrates that

1) Our techniques are efficient and can perform up 2 orders of magnitude better than the straightforward algorithm.
2) Both top-down and bottom-up strategies are scalable to the database size.
3) Overall, top-down strategy outperforms bottom-up strategy when the label distribution is relatively uniform. Moreover, the top-down strategy is more scalable as the graph size increases.
4) The bottom-up strategy outperforms the top-down strategy when the label distribution is biased.

VII. Related Work

Graph containment query has two subproblems. The first problem is subgraph containment query which has already been comprehensively studied. There are a lot of indexing and algorithms proposed for this problems. A major category is feature-based pruning, for example, GraphGrep [16], gIndex [2], TreePi [6], Tree-δ [4], FG-Index [7], and etc. Another category are non-feature-based techniques, namely, they are Closure-Tree [3], gString [17], GCoding [18], and [19] also proposed a tree-based graph decomposition technique. Recently, [5] proposes an efficient algorithm, QuickSI, to test the subgraph isomorphism between two graphs.

The above mentioned techniques are for exact subgraph containment query. A few similarity techniques have also been proposed, for example, Grafil [11], Closure-Tree [3], and etc.

In contrast to the subgraph containment problem, the supergraph containment problem receives much less attention. To the best of our knowledge, there only exist two previous works to solve the exact problem. However, they could not be applied to solve similarity supergraph containment problem.

In [8], Chen proposed a contrast subgraph-based indexing technique. The main idea is to capture the difference between database graphs and queries.

Zhang, in [9], uses a compact data structure to represent database graphs so that isomorphism test computation cost could be shared. They also proposed algorithms to mine important features from the database graphs.

The underlying problem, maximum common subgraph detection problem, is mostly investigated for *induced* subgraph only. Existing techniques fall into two categories, the maximal clique based paradigm [13], [14], [15] and the back-tracking paradigm [20], [21]. The maximal clique paradigm first constructs the association graph of the two given graphs and then detects the maximum clique of the association graph. Different from the maximum clique paradigm, the backtracking technique searches the maximum common subgraph by enumerate all common subgraphs of the two given graphs and choosing the largest one. Since it is well known that the detection of the maximum common subgraph is a NP-complete [1], many approximate algorithms have also been developed [22]. [23], [24] compared two such maximal clique based algorithms, Durand algorithm [14] and Balas-Yu algorithm [25], with a modification of McGregor's backtracking algorithm [21]. McGregor's algorithm is found to be up to 100 times more efficient for graphs with low connectivity, however it may be 104 times slower than the maximal clique algorithm on highly connected graphs. Note that these algorithms aim to find maximal common induced subgraph instead of common subgraph with the maximum number of edges.

VIII. CONCLUSION

In this paper, we studied the problem of similarity search on supergraph containment. We convert the underlying problem, maximum common subgraph detection, into σ-missing subgraph detection problem and propose a novel index-based algorithm, SigmaCSDetection. Two optimization methods have been proposed for databases with different label distributions. Since graph databases contain a large number of graphs, we propose global SG-Enum index to merge the local SG-Enum indexes into a global index. The global index not only saves the space but also reduces the query processing cost. Our techniques is up to two orders of magnitude faster than the straightforward solution in real datasets. As a possible future study, we will investigate how to extend our techniques to other types of graph queries, e.g., similarity search on subgraph containment.

ACKNOWLEDGMENT

The work was supported by ARC Grants (DP0987557, DP0881035 and DP0666428) and Google Research Award. The first and third authors' work was also partially supported by NICTA

REFERENCES

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

[2] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2004, pp. 335–346.

[3] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," in *Proceedings of the International Conference on Data Engineering*, 2006, p. 38.

[4] P. Zhao, J. X. Yu, and P. S. Yu, "Graph indexing: tree + delta <= graph," in *Proceedings of the International Conference on Very Large Data Bases*, 2007, pp. 938–949.

[5] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," in *Proceedings of the International Conference on Very Large Data Bases*, 2008, pp. 364–375.

[6] S. Zhang, M. Hu, and J. Yang, "Treepi: A novel graph indexing method," in *Proceedings of the International Conference on Data Engineering*, 2007, pp. 966–975.

[7] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2007, pp. 857–872.

[8] C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang, and X. Gu, "Towards graph containment search and indexing," in *Proceedings of the International Conference on Very Large Data Bases*, 2007, pp. 926–937.

[9] S. Zhang, J. Li, H. Gao, and Z. Zou, "A novel approach for efficient supergraph query processing on graph databases," in *Proceedings of the International Conference on Extending Database Technology*, 2009, pp. 204–215.

[10] K. S. Fu, "A step towards unification of syntactic and statistical pattern recognition," in *IEEE Trans. Pattern Anal. Mach. Intell.*, 1986.

[11] X. Yan, P. S. Yu, and J. Han, "Substructure similarity search in graph databases," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2005, pp. 766–777.

[12] J. W. Raymond and P. Willett, "Maximum common subgraph isomorphism algorithms for the matching of chemical structures." *J Comput Aided Mol Des*, vol. 16, no. 7, pp. 521–533, July 2002. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/12510884

[13] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, September 1973. [Online]. Available: http://dx.doi.org/10.1145/362342.362367

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

[15] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcolo*, vol. 9, no. 4, pp. 341–352, September 1972.

[16] R. Giugno and D. Shasha, "Graphgrep: A fast and universal method for querying graphs," *Proceedings of the International Conference on Pattern Recognition*, vol. 2, pp. 112–115 vol.2, 2002.

[17] H. Jiang, H. Wang, P. S. Yu, and S. Zhou, "Gstring: A novel approach for efficient search in graph databases," in *Proceedings of the International Conference on Data Engineering*, 2007, pp. 566–575.

[18] L. Zou, L. Chen, J. X. Yu, and Y. Lu, "A novel spectral coding in a large graph database," in *Proceedings of the International Conference on Extending Database Technology*, 2008.

[19] D. W. Williams, J. Huan, and W. Wang, "Graph database indexing using structured graph decomposition," in *Proceedings of the International Conference on Data Engineering*, 2007, pp. 976–985.

[20] E. B. Krissinel and K. Henrick, "Common subgraph isomorphism detection by backtracking search," *Softw. Pract. Exper.*, vol. 34, no. 6, pp. 591–607, 2004.

[21] J. J. McGregor, "Backtrack search algorithms and the maximal common subgraph problem," *Softw., Pract. Exper.*, vol. 12, no. 1, pp. 23–34, 1982.

[22] I. M. Bomze, M. Budinich, M. Pelillo, and C. Rossi, "Annealed replication: a new heuristic for the maximum clique problem," *Discrete Applied Mathematics*, vol. 121, no. 1-3, pp. 27–49, 2002.

[23] 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*, 2002, pp. 123–132.

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

[25] E. Balas and C. S. Yu, "Finding a maximum clique in an arbitrary graph," *SIAM J. Comput.*, vol. 15, no. 4, pp. 1054–1068, 1986.