FAST-ON*: AN EXTENDED ALGORITHM FOR GRAPH ISOMORPHISM PROBLEM AND GRAPH QUERY PROCESSING

Mosab Hassaan and Karam Gouda

Faculty of Computers and Informatics, Benha University, Egypt
{mosab.hassaan, karam.gouda}@fci.bu.edu.eg

ABSTRACT

Graphs are widely used to model complicated data semantics in many applications. In our paper [8], we proposed Fast-ON, an efficient algorithm for subgraph isomorphism problem. In this paper, we develop an efficient algorithm called Fast-ON that extends Fast-ON to handle two other problems, namely, graph isomorphism problem and graph query processing. Our performance study shows that Fast-ON* outperforms previously proposed algorithms of the two problems with a wide margin.*

KEYWORDS

graph isomorphism, graph query processing

1. INTRODUCTION

As a popular data structure, graphs have been used to model many complex data objects and their relationships in the real world, such as the chemical compounds [20], entities in images [16], and social networks [2], etc. For example, in social network, a person *i* corresponds to a vertex v_i in the graph G, and another person j corresponds to a vertex v_j in the graph G. If persons *i* and *j* are acquaintances or they have a business relation, then an edge (v_i, v_j) exists, which connects vertex v_i and v_j. Also in chemistry, a set of atoms combined with designated bonds are used to describe chemical molecules.

The **G**raph **I**somorphism **P**roblem (GIP) tests whether two given graphs are isomorphic. In other words, it asks whether there is a one-to-one mapping between the vertices of the graphs, preserving the edges. This problem has been studied for decades by mathematicians, chemists and computer scientists, and is considered interesting from both the theoretical and the practical point of view, since it has applications in many fields, ranging from pattern recognition and computer vision [7] to information retrieval [1], data mining [19], or chemistry [14]. For example, in data mining, one main challenges in frequent subgraph mining is to systematically generate candidate subgraphs in a non-redundant manner, such that we do not generate the same graph more than once. This means that we have to do graph isomorphism checking to make sure that duplicate graphs are removed. Given the two graphs q and q' in Figure 1, we have q is iosmorphic to q' (u_1 is mapped to u_3', u_2 is mapped to u_1', and u_3 is mapped to u_2'). GIP has not been possible thus far to prove it to be in the complexity class P nor NP-complete. Many algorithms have been proposed, such as Ullman [18], Nauty [15], and Vflib [4].

Graph query processing [6, 22, 3, 21, 10, 23, 9, 26, 17, 29, 11, 24, 28, 27, 25] has attracted much attention in recent years thanks to the increasing popularity of graph databases in various application domains. Existing research on graph query processing is conducted mainly on two types of graph databases as follows.

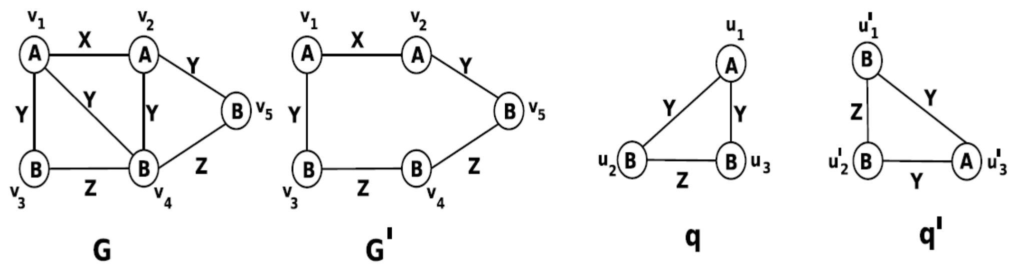


Figure 1: Running Example

The first one is a large graph such as social networks. Query processing on large graph (**S**ubgraph over **L**arge **G**raph (SLG)) can be described as follows. Given a query graph q and large graph G, we want to retrieve as output the set of subgraphs of G, each of which is isomorphic to q. For example, in protein-protein interaction networks, biologists may want to recognize groups of proteins which match a particular pattern in a large protein-protein interaction network. . Given the the graph *G* and and the query graph *q* in Figure 1, we have *q* is iosmorphic to two subgraphs in *G* (*the first one:* u_1 is mapped to v_1, u_2 is mapped to v_3, and u_3 is mapped to v_4 and *the second one:* u_1 is mapped to v_2, u_2 is mapped to v_4, and u_3 is mapped to v_5). There are many algorithms have been proposed for subgraph over large graph, such as [24, 28, 27, 25].

The second one is transaction graph databases that consist of a set of relatively smaller graphs. Transaction graph databases are prevalently used in scientific domains such as chemistry, bioinformatics, etc. Query processing on transaction graph databases (**S**ubgraph **S**earch **P**roblem (SSP)) can be described as follows. Given a query graph q and a graph database $D = \{g_1, g_2, g_3, ..., g_n\}$, we want to retrieve as output all graph $g_i \in D$ such that q is a subgraph of g_i. For example, given a large chemical compound database, a chemist may want to find all chemical compounds having a particular substructure. Given the databases graphs D = {*G, G'*} and query graph *q* in Figure 1, Graph *G* should be returned as the result since *G* contains *q*. There are many algorithms have been proposed for subgraph search problem, such as [6, 22, 3, 21, 10, 23, 9, 26, 17, 29, 11].

In this paper, we propose an efficient algorithm called Fast-ON* for testing GIP, SLG, and SSP. Fast-ON* is an extension of Fast-ON, a previously published algorithm developed by us [8]. We evaluate Fast-ON* and compare it with Ullman and Vflib on real and synthetic datasets for GIP, with SMS [27] and GADDI [24] for SLG, and with CT-index [11] and FG-index [3] for SSP.

Organization. This paper is organized as follows. Section 2 introduces the preliminary concepts. Section 3 presents the related work. Fast-ON* algorithm is introduced in Section 4. Section 5 reports the experimental results. We conclude in Section 6 by giving a summary and directions for future work. In Section 7 (Appendix), we give more details about Ullman and Fast-ON algorithms.

2. Preliminary Concepts

As a general data structure, labeled graph is used to model complex structured and schema-less data. In labeled graph, vertices and edges represent entity and relationship, respectively. The attributes associated with entities and relationships are called labels. This paper focuses on simple undirected graphs with vertex and edge labels or with vertex label only. Below, the terminology used throughout the paper is introduced.

Definition 2.1 Labeled Graph.

A labeled graph G is defined as a 4-tuples $< V_G, E_G, L_G, l_G >$, where V_G is the set of vertices, E_G is the set of edges, L_G is the set of labels, and l_G is a labeling function that maps each vertex or edge to a label in L_G.

Definition 2.2 Vertex Neighborhood.

Given a graph G, the neighborhood of $u \in V_G$ is the set $N_G(u) = \{v \in V_G \mid (u,v) \in E_G\}$. The degree of a vertex $v \in V_G$ is defined as $deg(v) = | N_G(v) |$ for simple graphs.

Definition 2.3 Graph Isomorphism Problem (SIP).

Given two graphs $H = < V_H, E_H, L_H, l_H >$ and $G = < V_G, E_G, L_G, l_G >$. A graph isomorphism from H to G is a bijection $f : V_H \mapsto V_G$ such that: (1) for any edge $(u,v) \in E_H$, there is an edge $(f(u), f(v)) \in E_G$, (2) $l_H(u) = l_G(f(u))$ and $l_H(v) = l_G(f(v))$, and (3) $l_H((u,v)) = l_G((f(u), f(v)))$.

The concept of **subgraph isomorphism probelm** can be defined analogously by using an *injection* instead of a *bijection*. A graph H is called a subgraph of another graph G (or G is a supergraph of H), denoted as $H \subseteq G$ (or $G \supseteq H$), if there exists a subgraph isomorphism from H to G.

Definition 2.4 Vertex Labeled Neighborhood. [8]

Given a graph G and a vertex $u \in V_G$, the labeled neighborhood of u is given as $NL_G(u) = \{(l_G(v), l_G((u,v))) : v \in V_G$ and $(u,v) \in E_G\}$.

The following theorem [8] presents the necessary condition required to map a vertex $u \in V_q$ to a vertex $v \in V_G$.

Theorem 2.1

Given two graphs q and G such that q is subgraph isomorphic G under injective function f. If $u \in V_q$ is mapped to $v \in V_G$, then $NL_q(u) \subseteq NL_G(v)$

Definition 2.5 Subgraph Over Large Graph (SLG).

Given a large data graph G and a query graph q, where $|V_q| << |V_G|$, the problem of subgraph over large graph is defined as to find all matches of q in G, where matches are defined in Definition 2.3.

Definition 2.6 Subgraph Search Problem (SSP).

Given a graph database $D = \{g_1, g_2, g_3, ..., g_n\}$ and a query graph q, the subgraph search problem is to find a set of graphs D_q which contain q from D, such that $D_q = \{g \mid g \in D \text{ and } q \subseteq G\}$.

3. Related Work

Ullman [18] and Vflib [4] are two well-known algorithms for (sub)graph isomorphism problem and Nauty [15] is well-known algorithm for graph isomorphism problem. Ullman algorithm is developed based on the branch and bound paradigm [13]. It is prohibitively expensive for querying against a very large data graph. The Vflib algorithm is another important algorithm for subgraph isomorphism problem. It uses an optimized serial version of Ullman algorithm. The algorithm proceeds by creating and modifying a match state. The match state contains a matched-set, which is a set of vertex pairs that match between the query graph *q* and data graph *G*. If the matched-set contains all of the query graph *q*, then the algorithm is successful and returns. Otherwise, the algorithm attempts to add a new pair. It does this by tracking the in-set and out-set of each graph, which are the sets of vertices immediately adjacent to the matched-set. These two sets define the potential vertices that can be added to a given state. The only pairs that can be added are either in the in-set of both graphs or the out-set of both graphs. The algorithm uses backtracking search to find either a successful match state, or return a failure. Nauty is a backtracking algorithm that traverses a search tree looking for a canonical labeling, and, in the process, builds the automorphism group of the graph. Nauty starts with an initial vertex classification by their degree, that defines a partition of the vertices. From this partition, it performs successive refinements based on the adjacencies of the vertices of a cell of the partition with the vertices in all the cells of the partition.

For subgraph over large graph (SLG), Ullman [18] and Vflib [4] cannot work well in large graphs. There are many algorithms have been proposed for SLG. GADDI [24] has been proposed for this problem. The authors of GADDI proposed an index based on Neighborhood discriminative substructures. It counts the number of small substructures in induced intersection graph between the neighborhood of two vertices. Nova [28] is another algorithm for SLG. Nova utilizes a noval index called nIndex. It pre-order the query vertices in a way such that more computational cost could be shared. It also employed an eagerly pruning strategy which could determine the current enumeration state is impossible to lead to a successful mapping, so that the enumeration process could exit early. Also SPath [25] has been proposed for this problem. SPath maintains for each vertex of the network (large graph) a neighborhood signature, a compact indexing structure comprising decomposed shortest path information within the vertex's vicinity. It revolutionizes the way of graph query processing from vertex-at-a-time to path-at-a-time. There is another algorithm called SMS [27] for SLG. It desgin vertex code based on the information of each vertex and its neighbors. The authors of SMS proposed the strategy of partitioning the large graph to improve the query performance.

For subgraph search problem (SSP), Ullman [18] and Vflib [4] also cannot work well since subgraph isomorphism problem is NP-complete problem [5] and we need to perform subgraph isomorphism checking of q against each graph $g_i \in D$. The major challenge in this scenario is to reduce the number of pairwise subgraph isomorphism checking. A number of graph indexing techniques have been proposed to address this challenge [22, 3, 23, 26, 17, 11, 10, 29]. There are two categories of graph indexing techniques: feature-based index and nonfeature-based index. In feature-based index [22, 3, 23, 26, 17, 11], some subgraphs are chosen as index features, and an inverted list is built for each feature. Generally, query processing follows the *filtering-and-verification* framework. for example, CT-index [11] is proposed for SSP. The authors of CT-index proposed new approach based on the *filtering-and-verification* framework, using a new hash-key fingerprint technique with a combination of tree and cycle features for filtering and a new subgraph isomorphism test for verification. In another new indexing technique FG-Index [3], both frequent subgraphs and infrequent edges are chosen as feature set and it supports verification-free strategy. In the category of non-feature-based index. Closure-tree [10] (a clustering-based index) has been proposed to support both subgraph search problem and similarity queries. The authors of Closure-tree proposed pseudo subgraph isomorphism using the strategy of checking the existence of semi-perfect matching between query graph and databases graphs. Also GCoding [29] has been proposed. Based on GCoding, the structure of the graph can be encoded into a numerical space, and a two-step filtering method is presented to search the graph database.

In Section 7 (Appendix), Since Fast-ON* is extended version of Fast-ON [8], we will give more details about Fast-ON algorithm and we will give also more details about Ullman algorithm [18].

4. FAST-ON* ALGORITHM

In this section, we present an algorithm called Fast-ON* that extends Fast-ON (see subsection 7.2. in Appendix) to handle two other problems, namely, graph isomorphism problem and graph query processing.

4.1. Graph Isomorphism Problem (GIP).

Considering the graph isomorphism instead of the subgraph isomorphism, we must apply the following. First, we test that the query graph and the data graph have the same number of vertices. If $|V_q|$ is not equal to $|V_G|$, we are sure that q is not isomorphic to G, thus the process could exit early. Then, we must modify the bit matrix M_{DLN} in Algorithm 5 (Subsection 7.2) as follows. $m(i, j) = 1$ if $DLN_q[i] = DLN_G[j]$, otherwise $m(i, j) = 0$. Fast-ON* algorithm for GIP is presented in the following algorithm.

Algorithm 1: $Fast\text{-}ON^*(q, G)$ for graph isomorphism problem (GIP)

```
Input: two graph  $q$ and $G$.
Output: Boolean: $q$ is isomorphic to $G$.
Boolean Test = FALSE; /* Global Variable */
1: if($|V_q| = |V_G|$) then
2:    $Fast\text{-}ON(q, G)$
3: else
4:    return FASLE
```

4.2. Graph Query Processing.

As we discussed in Section 1, graph query processing consists of two problems as follows.

4.2.1. Subgraph over Large Graph (SLG).

Recall, SLG can be described as follows. Given a query graph q and large graph G , we want to retrieve as output the set of subgraphs of G, each of which is isomorphic to q. Fast-ON * algorithm for SLG can be presented as follows. The output of this algorithm will be all subgraph somorphism mappings f of q against G. This will be done by some changes in Fast-ON algorithm (Algorithm 5) as follows. First, remove the statement Boolean Test = FALSE. Then replace the lines 7-9 (in Procedure: $Recursive\_Search(u_i)$) by only one line "Output a mapping f ". In some cases, we do not use distinct neighborhoods strategy (see optimization two in Subsection 7.2) since the large graph may have large size of distinct neighborhoods.

4.2.2. Subgraph Search Problem (SSP).

Recall, SSP can be described as follows. Given a query graph q and a graph database $D = \{g_1, g_2, g_3, ..., g_n\}$, we want to retrieve as output all graph $g_i \in D$ such that q is a subgraph of g_i. Fast-ON * algorithm for SSP is presented in the following algorithm.

```
------------------------------------------------------------------------------------
Algorithm 2: $Fast\text{-}ON^{*}(q, D)$ for subgraph search problem (SSP).
------------------------------------------------------------------------------------
Input: $q$ : a query graph and $D$ : a graph database.
Output: $D_q$ : the answer set.
1: $D_q = \phi$
2: for each $G \in D$ do
3:   Boolean Test = FALSE;
4:   if $Fast\text{-}ON(q, G)$
5:      $D_q = D_q \cup \{G\}$
------------------------------------------------------------------------------------
```

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of Fast-ON * on real and synthetic graphs. Fast-ON * is implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments were run on a PC with Intel 3GHz dual Core CPU and 4G memory running Linux. In experiments, we consider vertex-labeled and edge-labeled(or edge-unlabeled) simple undirected graphs.

5.1. Datasets

Experimental evaluation are performed on a group of real and synthetic datasets as follows.

Real Datasets. The first real dataset, referred to as AIDS_10K, consists of 10,000 graphs that are randomly drawn from the AIDS Antiviral screen database [1]. These graphs have 25 vertices and 27

[1] http://dtp.nci.gov/.

edges on average. There are totally 62 distinct vertex labels in the dataset but the majority of these labels are C, O and N. The total number of distinct edge labels is 3. The second real dataset, referred to as Chem_1M. it is a subset of the PubChem database [2], and consists of one million graphs. Chem_1M has 23.98 vertices and 25.76 edges on average. The number of distinct vertex and distinct edge labels are 81 and 3 respectively. For this study, we derive subsets from Chem_1M, each of which consists of N graphs and called Chem_N dataset. The third real dataset, referred to as HRPD, that is, a human protein interaction network (one large graph). It consists of 9,460 vertices, 37,000 edges and 307 generated vertex labels with GO term description. In this dataset, the edge labels and direction are ignored in our experiment.

Synthetic Datasets. The synthetic graph dataset is generated as follows: first, a set of S seed fragments (seed of a small subgraphs) is generated randomly, whose size is determined by a Poisson distribution with mean I. The size of each graph is a Poisson random variable with mean T. Seed fragments are then randomly selected and inserted into a graph one by one until the graph reaches its size. More details about the synthetic data generator are available in [12]. A typical dataset may have the following setting: it has 10,000 graphs and uses 100 seed fragments ($S = 100$) with distinct vertex labels, $L_V = 3$ and distinct edge labels, $L_E = 2$. On average, each graph has 50 edges ($T = 50$) and each seed fragment has 15 edges ($I = 15$). This dataset is denoted by Syn_10K.

Query Sets. For the three datasets AIDS_10K, Chem_1M and Syn_10K, there are six query sets *Q4, Q8, Q12, Q16, Q20* and *Q24*. Each set Qi consists of 1000 query graphs with i edges. For AIDS_10K, we adopt the query set from [22]. In order to generate query sets for Chem_1M and Syn_10K datasets, a set of 1000 graphs whose size larger than or equal to 24 are randomly selected from the dataset. Then, edges are removed from graphs such that the remaining graphs still connected. These graphs constitute Qi when all graphs are of size i. For HRPD dataset, we generate only three queries, namely, *q4*, *q8*, and *q12* with size 4 , 8, and 12 respectively.

5.2 Performance Study

In this section, we evaluate the performance of Fast-ON* on various datasets for the two problems, namely, graph isomorphism problem and graph query processing as follows.

5.2.1. Performance Study for Graph Isomorphism Problem (GIP)

In this subsection, for GIP, we evaluate the performance of Fast-ON* on the query sets of the three datasets, namely, AIDS_10k, Chem_1M, and Syn_10K against themselves, i.e., we perform Qi against Qi where $i \in \{4, 8, 12, 16, 20, 24\}$. This will be done by comparing Fast-ON* with the two algorithms Ullman and Vflib. Total response time in msec for each query set is recorded and demonstrated in Figure 2. From this Figure, Fast-ON* algorithm significantly outperforms Ullman algorithm and Vflib algorithm and it achieves even more performance gain with increasing query size.

5.2.2. Performance Study for graph query processing

In this subsection, we evaluate the performance of Fast-ON* for the two problems of graph query processing on real and synthetic graphs as follows.

1. *Performance Study for Subgraph over Large Graph (SLG).*

[2] ftp://ftp.ncbi.nlm.nih.gov/pubchem/.

For SLG, we evaluate the performance of Fast-ON* on HRPD dataset by comparing it with the two algorithms GADDI and SMS. Total response time in msec for the queries (*q4, q8,* and *q12*) is recorded and demonstrated in Figure 3. From this Figure, Fast-ON* algorithm significantly outperforms the two algorithms GADDI and SMS by 7 order of magnitude and a factor up to 2 respectively.

2. Performance Study for Subgraph Graph Search (SSP).

For SSP, we evaluate the performance of Fast-ON* on AIDS_40K and Chem_200K datasets by comparing it with the two algorithms FG-index and CT-index. Note that, the total response time of the algorithm FG-index (or CT-index) is equal to its index construction time plus its query processing time. Total response time in sec for each query set is recorded and demonstrated in Figure 4. In Figure 4(a), Fast-ON* algorithm significantly outperforms FG-index algorithm and CT-index algorithm by 5 order of magnitude and 2 order of magnitude respectively. In Figure 4(b), Fast-ON* algorithm significantly outperforms FG-index algorithm with a factor up to 4. Note that CT-index is not shown in Figure 4(b), since it failed to run on our machine.

6. CONCLUSION

In this paper, we develop an efficient algorithm called Fast-ON* that extends Fast-ON to handle two other problems, namely, graph isomorphism problem and graph query processing. The algorithm presented in this paper is very effective and efficient. The experimental results demonstrated that Fast-ON* outperforms the state-of-the-art algorithms of the two problems that studied in this paper on various datasets. Possible direction for future studies include an approximate graph query processing.

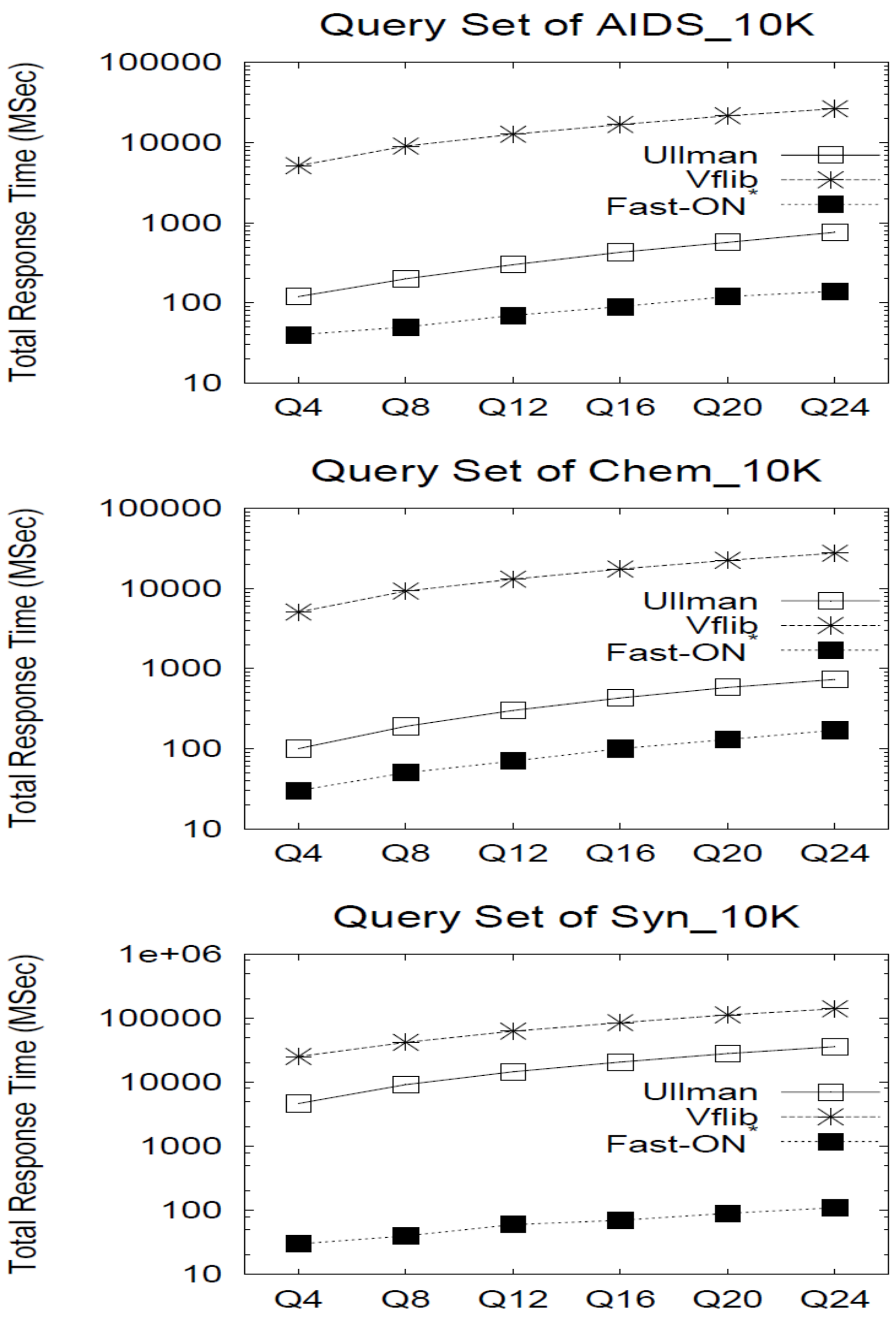


Figure 2: Performance Study for Graph Isomorphism Problem (GIP)

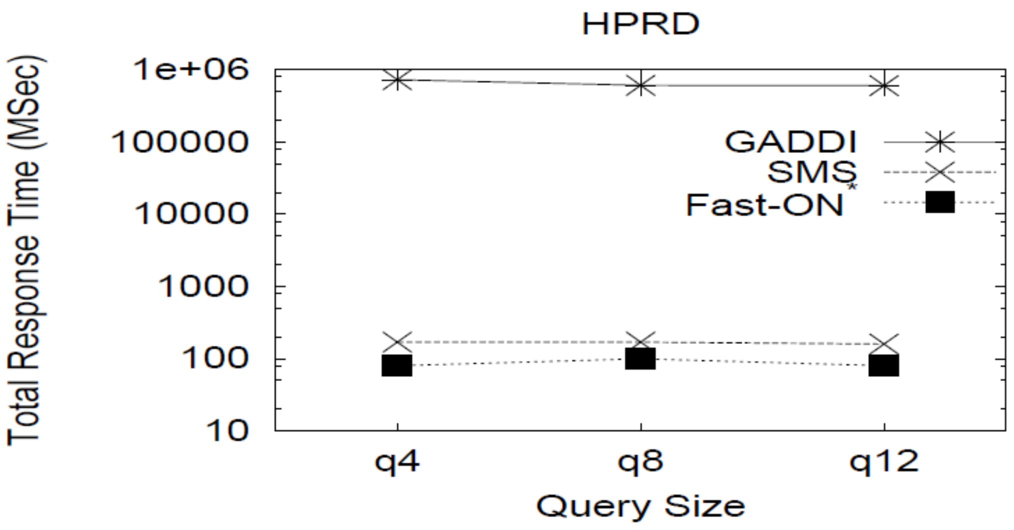
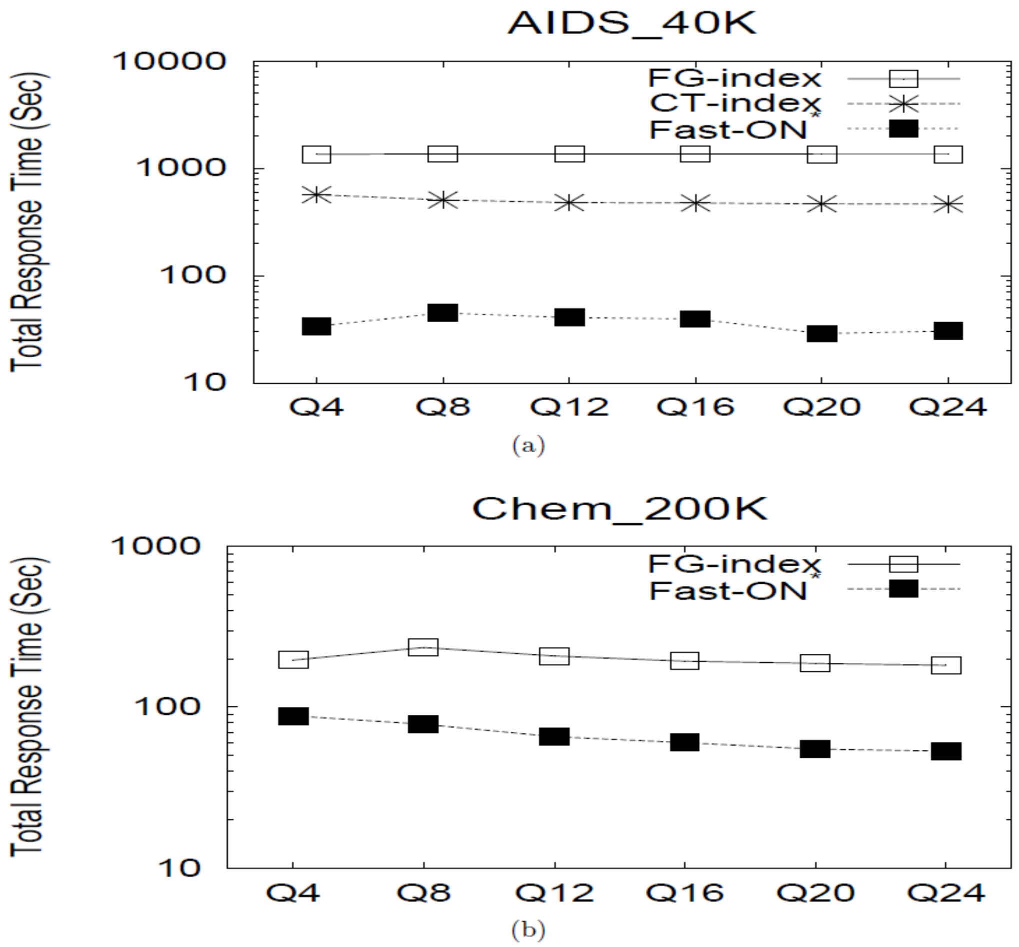


Figure 3: Performance Study for Subgraph over Large Graph (SLG)



F

igure 4: Performance Study for Subgraph Search Problem (SSP)

7. APPENDIX

In this section, we discuss Ullman algorithm and Fast-ON algorithm in more details as follows.

7.1. Ullman Algorithm [18]

Given a query graph *q* and a data graph *G*. To check if *q* is subgraph of *G*, Ullman's basic approach is to enumerate all possible mappings of vertices in V_q to those in V_G using a depth-first tree-search algorithm. Figure 5 shows a part of the search tree generated from testing the two graphs *G* and *q* (query graph) in Figure 1. At level *i* of the search tree, a vertex u_i in V_q is mapped to some vertex in V_G (the number *j* inside each node in the search tree means that this node represents the vertex $v_j \in V_G$). The root node of the search tree represents the starting point of the search, inner nodes of the search tree correspond to partial mappings, and nodes at level $|V_q|$ represent complete – not necessarily sub-isomorphic – mappings. If there exists a complete mapping that preserves adjacency in both *q* and *G*, then we have *q* is subgraph isomorphic to *G*, otherwise *q* is not subgraph isomorphic to *G*. The bold path in Figure 4, (u_1 is mapped to v_1, u_2 is mapped to v_3, and u_3 is mapped to v_4), is a complete mapping that preserves adjacency in *q* and *G*, thus *q* is subgraphs isomorphic to *G*.

Unfortunately, the number of complete mappings is exponential in the number of nodes of the involved graphs. This means that the running time may be huge even for reasonably small graphs. In order to cope with subgraph isomorphism problem efficiently, Ullman uses a refinement procedure to prune the search space. This procedure based on the following two conditions:

1. **Label and degree condition.** A vertex $u \in V_q$ can be mapped to $v \in V_G$ under injective mapping *f*, i.e., $v = f(u)$, if
(i) $l_q(u) = l_G(v)$, and
(ii) $deg(u) \le deg(v)$.

2. Neighbor condition. By this condition, Ullman algorithm examines the feasibility of mapping $u \in V_q$ to $v \in V_G$ by considering the preservation of structural connectivity. If there exist edges connecting *u* with previously explored vertices of *q* but there are no counterpart edges in *G*, the mapping test simply fails.

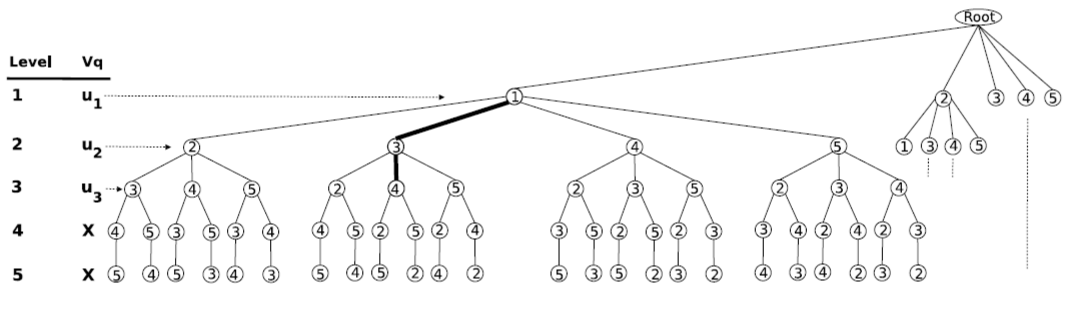


Figure 5: A Part of the Search Tree of Ullman Algorithm

7.2. Fast-ON Algorithm [8]

Our algorithm Fast-ON is based on Ullman algorithm. It improves Ullman algorithm by considering two effective optimizations as follows. It reduces the search space as much as possible by following a novel ordering strategy of the query's vertices (first optimization (Opt1)), and by utilizing the label information of vertex's neighborhood (second optimization (Opt2)). Comparing to Ullman [18] and Vflib [4], Fast-ON achieves up to 1-3 orders of magnitude speed-up. The two optimizations are explained as follows.

The first optimization is based on the observation that the search order in Ullman algorithm is random. Ullman algorithm depends on the order of query vertices imposed during input. This default ordering of V_q can possibly result in a search order that seriously slows down the algorithm. The adopted approach to order V_q is to require the currently processing query vertex to have high connectivity with the previously explored ones, that is, suppose that $u_i \in V_q$ is the currently processing vertex, then u_i should have the higher connectivity with $u_1, u_2, \ldots, u_{i-1}$ among the remaining ones. Whereas, u_1 is the one with maximum degree. This ordering forces unsuccesful mapping to be discarded as early as possible during the search, thus saving much of the time that Ullman algorithm may take on false long partial mappings. Algorithm 4 outlines this idea.

```
------------------------------------------------------------------------
Algorithm 4: $Order\_Vertices(V_q)$
------------------------------------------------------------------------
Input: $V_q = \{u_1, u_2, \ldots, u_{|V_q|}\}$;
Output: An order of $V_q$, $V_q' = \{u_1', u_2', \ldots, u_{|V_q|}'\}$;
1: $V_q' = \phi$;
2: for each $u \in V_q$ do calculate $deg(u)$;
3: $u_1' = u_k$, $k$ = argmax$_{u \in V_q}$ $deg(u)$;
4: Add $u_1'$ to $V_q'$ and remove $u_k$ from $V_q$;
5: for $i = 2 \ldots |V_q|$
6:     $u_1' = u_k$, $k$ = argmax$_{u \in V_q}$ $|\{(u,u') \in E_q : u' \in V_q'\}|$;
7:     Add $u_i'$ to $V_q'$ and remove $u_k$ from $V_q$;
8: return $V_q'$;
------------------------------------------------------------------------
```

For the second optimization, Fast-ON uses Theorem 2.1 to map a vertex $u \in V_q$ to a vertex $v \in V_G$ to improve the condition 2 in Ullman algorithm. To reduce the cost of the containment checks, Fast-ON cashing most of the repeated computation, as in the following steps:

1. Find the set of distinct labeled neighborhoods for the two graphs q and G, denoted as DLN_G and DLN_q respectively.

2. Construct a bit matrix $M_{DLN} = [m(i,j)]_{\alpha\beta}$ where $\alpha = |DLN_q|$ and $\beta = |DLN_G|$, to maintain the inclusion relationship between distinct neighborhoods of q and G, that is, $m(i,j) = 1$ if $DLN_q[i] \subseteq DLN_G[j]$, otherwise $m(i,j) = 0$.

3. For a graph g – g is q or G – construct an array of pointers P_g of size $|V_g|$, called position array, where each slot u holds the index of the vertex u labeled neighborhood at DLN_g.

Algorithm 5 outlines Fast-ON algorithm. Line 1 applies the first optimization Opt1, whereas lines 2-5 outline the second optimization Opt2. In line 5, for each query vertex $u \in V_q$, data graph vertices $v \in V_G$ that satisfy the modified first condition are collected into a set called candidate set $C(u)$. The procedure *Recursive_Search* matches u_i over $C(u_i)$ (line 5) and proceeds step-by-step by recursively matching the subsequent vertex u_{i+1} over $C(u_{i+1})$ (lines 6-7), or sets Test to true value and returns if every vertex of q has counterpart in G (line 9). If u_i exhausts all vertices in $C(u_i)$ and still cannot find matching, Recursive_Search backtracks to the previous state for further exploration (line 11). The procedure Matchable applies the second condition in [18]. This condition examines the feasibility of mapping $u \in V_q$ to $v \in V_G$ by considering the preservation of structural connectivity. If there exist edges connecting u with previously explored vertices of q but there are no counterpart edges in G, the mapping test simply fails.

```
------------------------------------------------------------------------------------------------------
Algorithm 5: $Fast-ON(q,G)$
------------------------------------------------------------------------------------------------------
Input: $q$ : a query graph and $G$ : a data graph.
Output: Boolean: $q$ is a subgraph of $G$.

Boolean Test = FALSE; /* Global Variable */

1: $V_q^{'} = Order\_Vertices(V_q)$;                /* Opt1 */
2: Construct $DLN_G, DLN_q$ and $M_{DLN}$;
3: Construct both $P_q$ and $P_G$;
4: for each $u \in V_q^{'}$ do
 5:     $C(u) = \{v : v \in V_G, l_q(u) = l_G(v), \text{ and } m(P_q(u), P_G(v)) = 1\}$; /* Opt2 (Cond. 1 of Ullman
                                                                                          after changing)*/
6: $Recursive\_Search(u_1)$;
7: return Test;
------------------------------------------------------------------------------------------------------
```

Algorithm 5: $Fast-ON(q,G)$ [continued]

```
Procedure Recursive_Search($u_i$)
1: if NOT Test then
2:   for $v \in C(u_i)$ and $v$ is unmatched do
3:     if NOT $Matchable(u_i, v)$ then continue;
4:     $f(u_i) = v$; $v$ = matched;
5:     if $i < |V_q^{'}|$ then
6:       Recursive_Search($u_{i+1}$);
7:     else
8:       Test = TRUE;
9:       return;
10:    $f(u_i)$ =NULL; $v$ = unmatched; /* Backtrack */

Function $Matchable(u_i, v)$ /* Cond. 2 of Ullman*/
1: for each $j < i$ do
2:   if $(v, f(u_j)) \notin E_G$ then return FALSE;
3: return TRUE;
```

REFERENCES

[1] A. T. Berztiss. A backtrack procedure for isomorphism of directed graphs. Journal of the ACM, Vol. 20, No. 3, pp365–377, 1973.

[2] D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. Proc. of PKDD, 2005.

[3] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. SIGMOD, pp857–872, 2007.

[4] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. IEEE transaction on pattern analysis and machine intelligence, Vol. 26, No. 10, pp1367–1372, 2004.

[5] M. R. Garey and D. S. Johnson. Computers and intractability; guide to the theory of NP-completeness. W. H. Freeman & Co., 1990.

[6] R. Giugno and D. Shasha. Graphgrep; a fast and universal method for quering graphs. Proc. of the 16th International Conference on Pattern Recognition, pp112–115, 2002.

[7] M. Gori, M. Maggini, and L. Sarti. Graph matching using random walks. Proc. of the 17th International Conference on Pattern Recognition, pp394–397, 2004.

[8] K. Gouad and M. Hassaan. A fast algorithm for subgraph search problem. Proc. of the 8th International Conference on Informatics and Systems, ppDE–53 – DE–59, 2012.

[9] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu. igraph: a framework for comparisons of disk-based graph indexing techniques. PVLDB, pp449–459, 2010.

[10] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. ICDE, pp38–49, 2006.

[11] Karsten Klein, Nils Kriege, and Petra Mutzel. Ct-index: Fingerprint-based graph indexing combining cycles and trees. ICDE, pp1115–1126, 2011.

[12] M. Kuramochi and G. Karypis. Frequent subgraph discovery. Proc. of ICDM, pp313–320, 2001.

[13] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. Econometrica, Vol. 28, No. 3, pp497–520.

[14] X. Liu and D. J. Klein. The graph isomorphism problem. Journal of Computational Chemistry, Vol. 12, No. 10, pp1243–1251, 1991.

[15] B. D. McKay. Practical graph isomorphism. Congressus Numerantium, Vol. 30, pp45–87, 1981.
[16] E. G. M. Petrakis and C. Faloutsos. Similarity searching in medical image databases. IEEE transactions on knowledge and data enginnering, , Vol. 9, No. 3, 1997.
[17] H. Shang, Y. Zhang, and X. Lin. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. PVLDB, pp364–375, 2008.
[18] J. R. Ullmann. An algorithm for subgraph isomorphism. ACM, , Vol. 23, No. 1, pp31–42, 1976.
[19] T. Washio and H. Motoda. State of the art of graph-based data mining. ACM SIGKDD Explorations Newsletter, , Vol. 5, No. 1, pp59–68, 2003.
[20] P. Willett. Chemical similarity searching. J. Chem. Inf. Computer Science, , Vol. 38, No. 6, 1998.
[21] D. W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. ICDE, pp976–985, 2007.
[22] X. Yan, S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. SIGMOD, pp335–346, 2004.
[23] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. ICDE, pp966–975, 2007.
[24] S. Zhang, S. Li, and J. X. Yu. Gaddi: distance index based subgraph matching in biological networks. EDBT, 2009.
[25] P. Zhao and J. Han. On graph query optimization in large networks. VLDB, 2010.
[26] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: tree + delta <= graph. VLDB, pp938–949, 2007.
[27] W. Zheng, L. Zou, and D. Zhao. Answering subgraph queries over large graphs. WAIM, pp390–402, 2011.
[28] K. Zhu, Y. Zhang, X. Zhu, and W. Wang. Nova: A novel and efficient framework for finding subgraph isomorphism mappings in large graphs. DASFAA, pp140–154, 2010.
[29] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A novel spectral coding in a large graph database. EDBT, pp181–192, 2008.