

SCALING SUBGRAPH MATCHING BY IMPROVING ULLMANN ALGORITHM

Karam GOUDA

Faculty of Computers and Artificial Intelligence

Benha University, Egypt

e-mail: karam.gouda@fci.bu.edu.eg

Gyöngyi BUJDOSÓ

Faculty of Informatics

University of Debrecen, Hungary

e-mail: bujdoso.gyongyi@inf.unideb.hu

Mosab HASSAAN

Faculty of Science

Benha University, Egypt

✉

Faculty of Informatics

University of Debrecen, Hungary

e-mail: mosab.hassaan@fsc.bu.edu.eg

Abstract. Graphs are vastly used to represent many complex data semantics in several domains. Subgraph isomorphism checking (an NP-complete problem) is a regular operation with this kind of data. In this paper, we propose an improvement of Ullmann algorithm, a well-known subgraph isomorphism checker. Our new algorithm is called Ullmann-ON^L. It utilizes a novel sorting method for query vertices and L -levels of vertex neighborhoods (N^L) to confine the search space of Ullmann algorithm. Our performance study shows that Ullmann-ON^L outperforms previously proposed algorithms with a wide margin.

Keywords: Subgraph matching, NP-complete, graph database

1 INTRODUCTION

Graphs have been used to represent a lot of complex objects and their relationships in the real world. Images entities [1], social networks [2], and chemical compounds [3] are data graphs examples. For instance, in social networks, a person x corresponds to a vertex w_x in the current network, and another person y corresponds to another vertex w_y in the current network. If x and y have a relation (business), then an edge $e = (w_x, w_y)$ will exist, which joins the two vertices w_x and w_y . Testing whether a graph contains another graph is considered as one of the main graph operations. In drug-discovery, for instance, given a large graph that represents a large chemical compound, a chemist may need to test if this large graph contains a small substructure or not. This test assists the designer of drugs to gain a primary view of the given substructure because these substructures have many similar biological activities. The containment testing operation is named as *subgraph isomorphism problem* and abbreviated as *SGI*. *SGI* is NP-complete [4]. Formally, a query graph g is a subgraph of another data graph H , if there is an injection from g vertices to H vertices which preserves the edge labels, the vertex labels, and the connections among vertices.

Figure 1 reports an example of *SGI*, where a query graph g and a data graph H are listed. The character beside each vertex represents its index and the character inside the vertex represents its label, and the character through the edge represents its label. Here, H contains g because the g vertices w_1 , w_2 , and w_3 can be mapped to the H vertices z_1 , z_3 , and z_4 , respectively. In this mapping, we preserve the vertex labels, the edge labels and the connections among vertices. Unfortunately, *SGI* is very difficult since it may require testing many of the injection candidates for labels and connections preservation before catching the correct one.

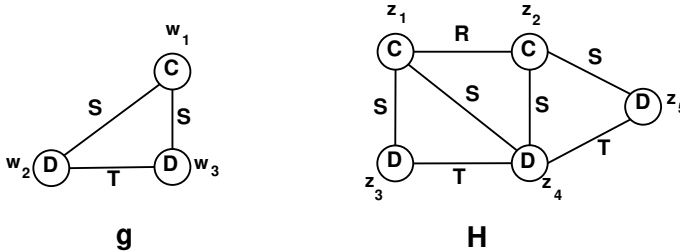


Figure 1. H: a data graph and g: a query graph

Our Contributions. In this paper, we propose an efficient subgraph isomorphism checking algorithm. It is based on Ullmann algorithm. Here, the search space is sharply reduced by following a novel sorting method of the query vertices, and by utilizing the label information at different levels of vertex's neighborhood. The new algorithm is called Ullmann-ON^L, where -ON^L stands for the bold letters in:

Subgraph checking by efficient sOrting the query’s vertices and applying the labeled information of **L**-levels of vertex’s Neighborhood. A preliminary version of this paper is listed in [5]. This paper extends the preliminary version by generalizing the neighborhood concept. Extra experiments are conducted to further assess Ullmann-ON^L. These experiments are performed to evaluate the effect of the new neighborhood concept on reducing the search space. Comparing to the well-known basic algorithms Ullmann [6] and Vflib [7], Ullmann-ON^L achieves up to 1-3 orders of magnitude and outperforms the recent QuickSI [8] by more than a factor four. Another experiment also shows that Ullmann-ON^L performs well on large data graph, by comparing it with vflib3 and DAF. In this experiment, we found that DAF is the worst whereas Ullmann-ON^L and vflib3 have similar performance.

The remainder of this paper is organized as follows. Section 2 represents the related work. Section 3 discusses the preliminary concepts. Section 4 reviews the Ullmann algorithm in details. Section 5 presents our algorithm, Ullmann-ON^L. The experimental results are reported in Section 6. Finally, the paper is concluded in Section 7.

2 RELATED WORK

Ullmann [6], Vflib2 [7], and QuickSI [8] are well-known algorithms for the subgraph isomorphism problem. Ullmann is based on the branch and bound paradigm [9]. It has poor performance for querying against the huge data graph. More details about Ullmann algorithm are given in Section 4.

Vflib2, on the other hand, utilizes an optimal version of Ullmann. It proceeds by constructing and changing a match state. The match state consists of a matched-set. This matched-set is a vertex pairs set matching between the two graphs, query graph g and the data graph H . If the matched-set consists of all the query vertices, then g is subgraph isomorphic to H and returns. Otherwise, the algorithm tries to insert a new vertex pair. This is done by tracking the in(out)-set of each graph, that are the vertices sets immediately connected to the matched-set. These two sets select the optimal vertices that can be inserted to a given state. The only pairs that can be inserted are either in the in-sets or in the out-sets of the two graphs. The algorithm also utilizes backtracking to search for either a successful match state, or terminate. Vflib3 [10] is an extension of vflib2 which leverages more pruning rules such as matching order and vertex classification. QuickSI significantly improves Ullmann by using the label and edge frequencies in the data graph to determine an effective search order of the search space.

Spath [11], TurboIso [12], GADDI [13], BoostIso [14], and DAF [15] are recently proposed algorithms for *SGI*. In Spath [11], paths are used as matching entities and neighborhood signatures of vertices are used to minimize the search space. TurboIso [12] combines similar query vertices and BoostIso [14] generalizes the idea of TurboIso to the data graphs.

GADDI [13] proposed a novel structure distance based approach to search matches of the running query graph in efficient way. Based on DAG ordering, DAF [15] adopted adaptive matching order and failing sets were used to enhance the pruning power.

In [16], a fair experimental study is conducted for the state-of-the-art subgraph isomorphism algorithms. The study revealed that vflib3, QuickSI, and DAF are the best three subgraph matching algorithms.

3 PRELIMINARY CONCEPTS

This paper focuses on simple, undirected graphs with vertex labels and edge labels. The terminology in this paper is presented as follows.

Definition 1 (Labeled Graph). A labeled graph H is a 4-tuple $\langle V_H, E_H, L_H, l_H \rangle$, where V_H is the vertex set, E_H is the edge set, L_H is the label set, and l_H is a labeling function that maps each vertex or edge to a label in L_H .

The number of edges in graph H is called H size.

Definition 2 (Labeled Path). A labeled path from a vertex w_1 to a vertex w_k in a labeled graph H is a sequence of vertex and edge labels in the following order: $l(w_1)l((w_1, w_2))l(w_2)l((w_2, w_3))l(w_3) \dots l(w_{k-1})l((w_{k-1}, w_k))l(w_k)$, where $(w_{i-1}, w_i) \in E_H$ is an edge in the graph.

Definition 3 (Vertex Neighborhood). Given a graph H , the neighborhood of vertex $w \in V_H$ is the set $N_H(w) = \{x \in V_H \mid (w, x) \in E_H\}$.

Note that the degree of a vertex $w \in V_H$ denoted as $\deg(w)$ is the size of its neighborhood, i.e. $\deg(w) = |N_H(w)|$.

Definition 4 (Subgraph Isomorphism). Given two graphs $X = \langle V_X, E_X, L_X, l_X \rangle$ and $Y = \langle V_Y, E_Y, L_Y, l_Y \rangle$. A subgraph isomorphism from Y to X ($Y \subseteq X$) is injection mapping $f : V_Y \mapsto V_X$ that satisfying the following three conditions:

- $\forall (w_1, w_2) \in E_Y$, there is an edge $(f(w_1), f(w_2)) \in E_X$,
- $l_Y(w_1) = l_X(f(w_1))$ and $l_Y(w_2) = l_X(f(w_2))$,
- $l_Y((w_1, w_2)) = l_X((f(w_1), f(w_2)))$.

The definition of *graph isomorphism* is as in Definition 4 except we use the bijection mapping instead of the injection mapping.

There may exist several subgraph isomorphisms from Y to X . Each represents an embedding (or occurrence) of the subgraph Y in X . A *graph automorphism* is an isomorphism mapping from a graph to itself. Two occurrences are *redundant* if their corresponding subgraphs are automorphic.

	w_1	w_2	w_3
f_1	z_1	z_3	z_4
f_2	z_1	z_4	z_3
f_3	z_2	z_4	z_5
f_4	z_2	z_5	z_4

Table 1. All possible subgraph isomorphisms (all embeddings)

Example 1. Consider the two graphs g and H in Figure 1. Here, g is a subgraph isomorphic to H based on the mapping f with $f(w_1) = z_1$, $f(w_2) = z_3$ and $f(w_3) = z_4$. In Table 1, all possible subgraph isomorphisms from g to H (all embeddings) are listed. The subgraph isomorphisms f_1 and f_2 represent two redundant embeddings. This also occurs with f_3 and f_4 .

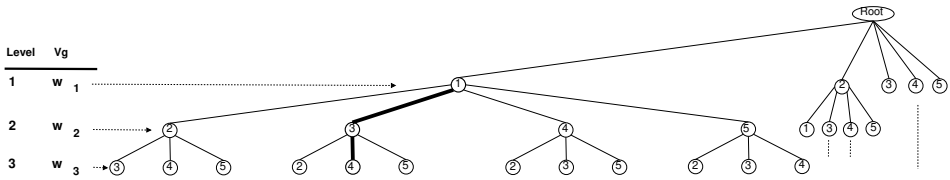


Figure 2. A subtree search of the Ullmann algorithm

The set of all symbols used through this paper is reported in Table 2

4 ULLMANN ALGORITHM

Ullmann algorithm is the earliest and known algorithm for *SGI*. Given a query graph g and a data graph H . To test if g is subgraph isomorphic to H , Ullmann algorithm enumerates all possible mappings from g vertices (V_g) to H vertices (V_H) using a depth-first-tree-search. Figure 2 reports a part of the search tree generated by checking the two graphs g and H in Figure 1. At each level l of this search tree, a vertex w_l in V_g is mapped to some vertex in V_H (in the tree, the variable k inside each vertex represents its *id* in the data graph H). A complete mapping is represented by a path from the root of the search tree to the last level ($|V_g|$). If there exists a complete mapping preserving the connections in the two graphs g and H , then g is subgraph isomorphic to H , otherwise not. In Figure 2, the bold path ($f(u_1) = z_1$, $f(u_2) = z_3$, and $f(u_3) = z_4$) is a complete mapping preserving the connections in the two graphs g and H . Therefore, g is subgraph isomorphic to H .

Note that, in Ullmann algorithm, the complete mappings count is exponential with respect to the vertices of the running query and data graphs. Thus the to-

Symbol	Description
g and H	Two graphs (the running query graph and the running data graph, respectively)
V_X and E_X	The set of vertices and edges of a running graph X , respectively.
$\deg(w)$	The degree of vertex w that belongs to the vertices set of running graph.
$N_X(w)$	The neighborhood of vertex $w \in V_X$.
f	The mapping between two sets of vertices (for instance, V_X and V_Y) such that f is injection mapping.
L_X	The label set of running graph X that maps each vertex $w \in V_X$ (or each edge $e \in E_X$) to vertex label, $l(w)$ (or edge label, $l(e)$).
\subseteq	Subgraph isomorphism between two running graphs.
V'_g	Efficient sorting of V_g based on the first optimization. Note that, g is the running query graph.
L	The neighborhood level which equals to one or two.
$N^L_H(w)$	L -scope Labeled Neighborhood of a vertex $w \in V_H$.
$Cand^0(w)$	The naive matching candidate set of vertex $w \in V_g$ where g is the running query graph.
$Cand^L(w)$	The neighborhood matching candidate set of vertex $w \in V_g$ where g is the running query graph.
DN^L_X	The distinct labeled neighborhoods set of running graph X .
M_{DN^L}	A bit matrix with size $ DN^L_g = \alpha^L x \beta^L = DN^L_H $ which based on the containment between DN^L_g and DN^L_H .
P^L_X	A position array with size equals to $ V_X $. For each vertex $w \in V_X$, the index of its labeled neighborhood is stored in position w .

Table 2. All symbols used through the paper

tal response time will be increased even for some small graphs. To address these challenges of *SGI*, Ullmann proposed a refinement procedure. Using the refinement procedure, the search space is pruned as much as possible. More details about refinement procedure are discussed by a three conditions as follows:

Condition I (Checking Label and Degree). A vertex $w \in V_g$ is mapped to a vertex $z \in V_H$ via the injection mapping f , i.e. $z = f(w)$, if

1. $l_g(w) = l_H(z)$,
2. $\deg(w) \leq \deg(z)$.

Condition II (Checking One-to-One Mapping). If $f(w) = z$ where $w \in V_g$ and $z \in V_H$ then we must not map any vertex $w' \in V_g$ to $z \in V_H$.

Condition III (Checking Neighbor). Here, Ullmann checks the feasibility of the mapping $w \in V_g$ to $z \in V_H$ by taking into account the preservation of structural connectivity. If there exist edges connecting w with previously explored vertices of the query graph g and at the same time there are no counterpart edges in H , the mapping check is failed.

Based on the first condition, the naive matching candidate set of query vertices is defined as follows.

Definition 5 (Naive Matching Candidate Set). Given a query g and a data graph H . Let $w \in V_g$ be a query vertex, the naive matching candidates set of w , based on the label and degree condition, is given as follows: $Cand^0(w) = \{x \in V_H : l_g(w) = l_H(x) \text{ and } \deg(w) \leq \deg(x)\}$.

Thus in Ullmann algorithm, for each vertex $w \in V_g$, an exhaustive search of all possible one-to-one correspondences to $z \in Cand^0(w)$ is searched. Thus, Ullmann search space is $\prod_{i=1}^P Cand^0(w_i)$, where $P = |V_g|$. The worst-case of the time complexity is $O(T^P)$, where $T = |V_H|$ and $P = |V_g|$. This is a result of the NP-completeness of *SGI*.

5 ULLMANN-ON^L

In this section, we propose Ullmann-ON^L, a novel algorithm for *SGI*. Ullmann-ON^L is based on Ullmann. Note that the search space of Ullmann is huge even after applying the refinement procedure. To address this problem, Ullmann-ON^L uses two novel optimizations to reduce the search space as much as possible as follows.

Algorithm: *Sort_Vertices*(V_g)

Input: $V_g = \{w_1, w_2, \dots, w_{|V_g|}\}$

Output: An sorting of V_g , $V'_g = \{w'_1, w'_2, \dots, w'_{|V_g|}\}$

```

1   $V'_g = \phi$ 
2  for each  $w \in V_g$ 
3    Compute  $\deg(w)$ 
4   $w'_1 = w_l, l = \arg \max_{w \in V_g} \deg(w)$ 
5  Insert  $w'_1$  in  $V'_g$  and delete  $w_l$  from  $V_g$ 
6  for  $k = 2 \dots |V_g|$ 
7     $w'_k = w_m, m = \arg \max_{w \in V_g} |\{(w, w') \in E_g : w' \in V'_g\}|$ 
8    Insert  $w'_k$  in  $V'_g$  and delete  $w_k$  from  $V_g$ 
9  return  $V'_g$ 

```

Figure 3. Sorting query vertices algorithm

5.1 Opt1: Sorting the Query Vertices

In Ullmann, the search order of query vertices is random. This default sorting of V_g can possibly result in a search order that seriously slows down the Ullmann as the connectivity between consecutive query vertices could be lost, which requires the Ullmann to consume a lot of time testing the feasibility of the partial mappings.

Query vertices must be explored in the sort which facilitates obtaining the all benefits of using the neighbor condition. The sorting of query vertices requires that the current processing vertex must have a high connections with the query vertices which previously explored. In other words, if $w_l \in V_g$ is the current processing vertex then w_l must have a higher connections with the vertices w_1, w_2, \dots, w_{l-1} in V_g . This novel sorting method discards the false mappings as early as possible during the search. Therefore, we save a lot of time that Ullmann takes on searching the false long partial mappings. The algorithm in Figure 3 presents this optimization.

5.2 Opt2: Utilizing Different Levels of Vertex Neighborhood

In this optimization, we propose a new condition based on the neighborhood labels of matching vertices. By this condition, the search space is reduced by minimizing $|Cand^0(u)|$ for each query vertex $w \in V_g$. Note that this condition is more effective than the label and degree condition in the Ullmann. See the following for more details.

For each vertex, we define its L -size labeled path and its L -scope labeled neighborhood, for a non-negative integer L , as follows.

Definition 6 (L -size labeled path of a vertex). The L -size labeled path of a vertex u is a simple, labeled path of size L , starting or ending at u .

Definition 7 (L -scope Labeled Neighborhood of a vertex). Given a graph G and a vertex $u \in V_G$, the L -scope labeled neighborhood of u , denoted $N_G^L(u)$, is the set of L -size labeled paths of u . Here L is also called the neighborhood level.

Next theorem reports the necessary condition required for mapping a vertex $w \in V_g$ to a vertex $z \in V_H$.

Theorem 1. Given the two graphs g and H such that g is subgraph isomorphic to H via the injection mapping f . If $w \in V_g$ is mapped to $z \in V_H$, then $N_g^L(w) \subseteq N_H^L(z)$.

According to Theorem 1, if the labeled neighborhood of vertex $z \in V_H$ does not include the labeled neighborhood of vertex $w \in V_g$, w cannot be mapped to z . By this containment test, we reduce the search space as much as possible. The following condition generalizes the first condition of Ullmann by applying this containment test rather than the degree test.

Label and neighborhood containment condition. A vertex $w \in V_g$ can be mapped to $z \in V_H$ using the injection mapping f , i.e. $z = f(w)$, if

1. $l_g(w) = l_H(z)$,
2. $N_g^L(w) \subseteq N_H^L(z)$.

Note that if we set $L = 1$ and $N_g^1(w) \subseteq N_H^1(z)$ is satisfied, it directly leads to $\deg(w) = |N_g^1(w)| \leq |N_H^1(z)| = \deg(z)$.

Example 2. Given the query graph g and the data graph H in Figure 1. According to the label and neighborhood containment condition and let $L = 1$, we can map vertex $w_1 \in V_g$ to $z_1 \in V_H$ since

1. $l_g(w_1) = l_H(z_1) = C$, and
2. $N_g^1(w_1) = \{(C, S, D), (C, S, D)\} \subseteq \{(C, R, C), (C, S, D), (C, S, D)\} = N_H^1(z_1)$.

Based on Theorem 1 we have the next definition.

Definition 8 (Neighborhood Matching Candidate Set). Given the query graph g and the data graph H . Assume that there is a vertex $w \in V_g$. The neighborhood matching candidate set of w is $Cand^L(w) = \{z \in V_H : N_g^L(w) \subseteq N_H^L(z)\}$ with $L = 1$ or 2 .

Next example shows the search space size when using the naive matching candidate set, $Cand^0$ and the neighborhood matching candidate set $Cand^L$ where $L = 1$ or 2 .

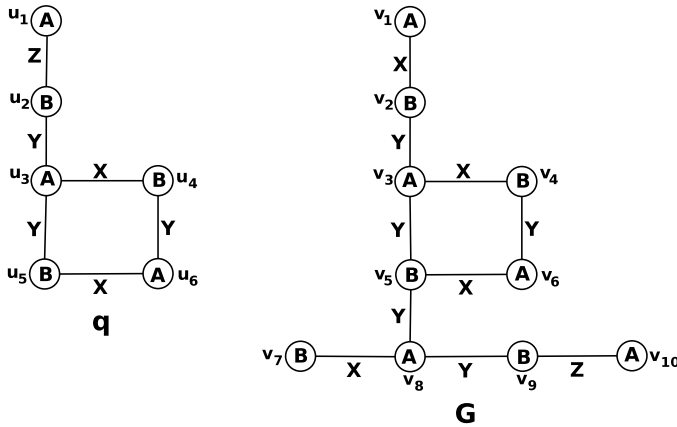


Figure 4. Running example (Opt2: utilizing neighborhood labels)

Example 3. Given the query graph q and the data graph G in Figure 4. We have three cases of the matching candidate set.

1. The first case if we use the naive matching candidate set, $Cand^0$ (Ullmann). Then $Cand^0(u_1) = \{v_1, v_3, v_6, v_8, v_{10}\}$, $Cand^0(u_2) = \{v_2, v_4, v_5, v_9\}$, $Cand^0(u_3) = \{v_3, v_8\}$, $Cand^0(u_4) = \{v_2, v_4, v_5, v_9\}$, $Cand^0(u_5) = \{v_2, v_4, v_5, v_9\}$, and $Cand^0(u_6) = \{v_3, v_6, v_8\}$.

The search space size is as follows:

$$\prod_{i=1}^{|V(q)|} |Cand^0(u_i)| = \prod_{i=1}^6 |Cand^0(u_i)| = 5 \times 4 \times 2 \times 4 \times 4 \times 3 = 1920.$$

u in $V(q)$	$Cand^1(u)$

Figure 5. The candidate vertices based on N_G^1 , $Cand^1$

- The second case if we use the neighborhood matching candidate set and let $L = 1$, $Cand^1$ (Ullmann-ON¹). Then $Cand^1(u_1) = \{v_{10}\}$, $Cand^1(u_2) = \{v_9\}$, $Cand^1(u_3) = \{v_3, v_8\}$, $Cand^1(u_4) = \{v_2, v_4, v_5\}$, $Cand^1(u_5) = \{v_2, v_4, v_5\}$, and $Cand^1(u_6) = \{v_3, v_6, v_8\}$. See Figure 5.

The search space size is as follows:

$$\prod_{i=1}^{|V(q)|} |Cand^1(u_i)| = \prod_{i=1}^6 |Cand^1(u_i)| = 1 \times 1 \times 2 \times 3 \times 3 \times 3 = 54.$$

- The final case if we use the neighborhood matching candidate vertices and let $L = 2$, $Cand^2$ (Ullmann-ON²). Then $Cand^2(u_1) = \{v_{10}\}$, $Cand^2(u_2) = \{v_9\}$, $Cand^2(u_3) = \phi$, $Cand^2(u_4) = \{v_4\}$, $Cand^2(u_5) = \{v_5\}$, and $Cand^2(u_6) = \{v_3, v_6\}$. See Figure 6.

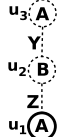

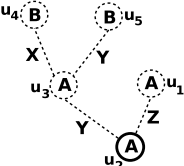
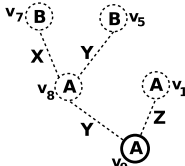
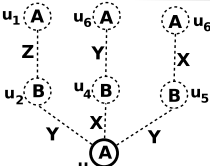
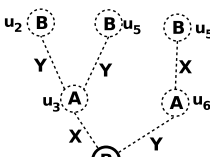
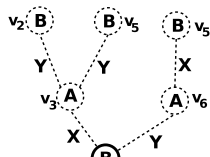
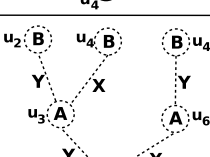
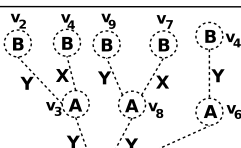
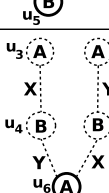
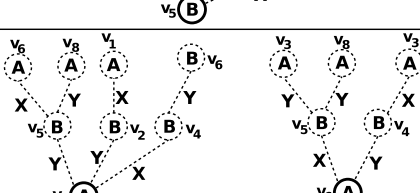
u in V(q)	Cand ² (u)
	
	
	<p style="text-align: center;">Phi</p>
	
	
	

Figure 6. The candidate vertices based on N_G^2 , $Cand^2$

The search space size is as follows:

$$\prod_{i=1}^{|V(g)|} |Cand^2(u_i)| = \prod_{i=1}^6 |Cand^2(u_i)| = 1 \times 1 \times 0 \times 1 \times 1 \times 2 = 0.$$

From the previous example, Ullmann-ON^L algorithm cuts down the search space by using the new vertex labeled neighborhood. The best case here is the final case that we used $L = 2$ ($Cand^2$) i.e. $|Cand^2| < |Cand^1| < |Cand^0|$. In experimental evaluation section we test the size of $Cand^0$ (Ullmann), $Cand^1$ (Ullmann-ON¹, which applies the first level of the neighborhood), $Cand^2$ (Ullmann-ON², which applies the second level of the neighborhood), and $Cand^{1,2}$ (Ullmann-ON^{1,2}, which applies the first and the second levels of the neighborhood together) on different datasets.

Recall, we reduce the search space as much as possible using the label and neighborhood containment condition. Unfortunately, computing the containment test is very expensive especially for large and dense graphs. Therefore, we propose a novel efficient method to compute the containment test. Our novel method is based on the following observation. In real graphs, a lot of vertices in these graphs share a similar neighborhood. Next example demonstrates this observation at $L = 1$.

Example 4. Assume, we have the query graph g and the data graph H given in Figure 1 and $L = 1$. We have the following:

1. In the data graph $H : N_H^1(z_1) = N_H^1(z_2) = \{(C, R, C), (C, S, D), (C, S, D)\}$, $N_H^1(z_3) = N_H^1(z_5) = \{(D, S, C), (D, T, D)\}$, and $N_H^1(z_4) = \{(D, S, C), (D, S, C), (D, T, D), (D, T, D)\}$.
2. In the query graph $g : N_g^1(w_1) = \{(C, S, D), (C, S, D)\}$, and $N_g^1(w_2) = N_g^1(w_3) = \{(D, S, C), (D, T, D)\}$.

Also, the previous observation occurs at $L = 2$. Therefore, we will minimize the cost of the containment tests by caching many of duplicated computations as follows:

- Compute the distinct labeled neighborhoods set for the two graphs g and H , namely DN_g^L and DN_H^L , respectively.
- Construct a bit matrix $M_{DN^L} = (m_{ij}^L)_{\alpha^L, \beta^L}$ where $\alpha^L = |DN_g^L|$ and $\beta^L = |DN_H^L|$, to maintain the containment relationship between distinct neighborhoods of g and H , with $m_{ij}^L = 1$ if $DN_g^L[i] \subseteq DN_H^L[j]$, otherwise $m_{ij}^L = 0$.
- For a graph X (query or data graph), construct a pointers array P_X^L with size $|V_X|$. This array is called position array. Each slot t in this array holds the id of the vertex t labeled neighborhood at DN_X^L .

Therefore, for each $w \in V_g$ and $z \in V_H$, we have the following:

$$N_g^L(u) \subseteq N_H^L(v) \quad \text{iff} \quad m_{P_g^L(u)P_H^L(v)} = 1.$$

Then, we can replace the test 2. in label and neighborhood containment condition by the following:

$$m_{P_g^L(u)P_H^L(v)} = 1.$$

5.3 Ullmann-ON^L Algorithm

Figure 7 reports the Ullmann-ON^L algorithm. Line 1 outlines Opt1 (the first optimization), whereas lines 2–5 apply Opt2 (the second optimization). In line 5, for each query vertex $w \in V_g$, the vertices of data graph $z \in V_H$ that satisfy the first condition (modified one) are inserted into $Cand^L(w)$ (the candidate set of the vertex w). The procedure *Recursion_Search* matches w_i over $Cand^L(w_i)$ (line 5) and proceeds step-by-step by recursively matching the subsequent vertex w_{i+1} over $Cand^L(w_{i+1})$ (lines 6–7), or sets the value of Flag to TRUE and returns if every vertex of g has its counterpart in H (line 10). If w_i exhausts all vertices in $Cand^L(w_i)$ and still cannot find matching, then the procedure *Recursion_Search* backtracks to the previous state for further exploration (line 11). The function *Joinable* outlines the condition 3. This function outputs the first embedding only. To output all embeddings, we can delete the lines 1 and 9 in the procedure *Recursion_Search*. Based on the Optimization 2, for each query vertex w , the candidate set of w ($Cand^L(w)$) is sharply minimized. Moreover, based on Optimization 1, false mappings are removed in early state. This saves more of required computations that Ullmann spent.

6 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of Ullmann-ON^L on real and synthetic graphs. Ullmann-ON^L is implemented in C++ with STL and compiled with GNU GCC. Experiments were run on a PC with Intel 3GHz dual Core CPU and 4GB memory running Linux.

In the following experiments, we consider vertex-labeled and edge-(un)labeled simple graphs.

6.1 Datasets

Experimental evaluation are performed on a group of real and synthetic datasets as follows. There are three real datasets (AIDS_40K, Chem_1M, and HRPD) and one synthetic dataset (Syn_10K). The corresponding information of these real and synthetic datasets is summarized in Table 3, where $|\mathcal{D}|$ represents the number of graphs, Avg. Vertex and Avg. Edge are the average vertex size and the average edge size, respectively. While Dist. Vertex Lab. and Dist. Edge Lab. denote the distinct vertex and edge labels, respectively.

¹ <http://dtp.nci.gov/>

² <ftp://ftp.ncbi.nlm.nih.gov/pubchem/>

Algorithm: Ullmann-ON^L

Input: A query graph (g), a data graph (H), and the neighborhood level (L).
Output: TRUE or FALSE: g is a subgraph isomorphic to H ?

Flag = FALSE; // Boolean Global Variable

- 1 $V'_g = \text{Sort_Vertices}(V_g)$ // First Opt.
- 2 **Compute** DN_g^L , DN_H^L , and M_{DN^L}
- 3 **Compute** the two arrays P_q^L and P_H^L
- 4 **for each** $w \in V'_g$ **do**
- 5 $Cand^L(w) = \{z : z \in V_H, l_g(w) = l_H(z), \text{ and } m_{P_q^L(w)P_H^L(z)}^L = 1\}$ // Second Opt.
- 6 $\text{Recursion_Search}(w_1)$
- 7 **return** Flag

Procedure $\text{Recursion_Search}(w_i)$

- 1 **if NOT** Flag **then**
- 2 **for** $z \in Cand^L(w_i)$ and z is unmatched **do** // Condition II
- 3 **if NOT** $\text{Joinable}(w_i, z)$
- 4 **continue**;
- 5 $f(w_i) = z$ i.e. z is matched
- 6 **if** $i < |V'_g|$
- 7 $\text{Recursion_Search}(w_{i+1})$
- 8 **else**
- 9 Flag = TRUE;
- 10 **return**;
- 11 $f(w_i) = \text{NULL}$ i.e. z is unmatched // Backtracking

Function $\text{Joinable}(w_i, z)$ // Condition III

- 1 **for each** $(w_i, w_j) \in E_g, j < i$ **do**
- 2 **if** $(z, f(w_j)) \notin E_H$
- 3 **return** FALSE
- 4 **return** TRUE

Figure 7. Ullmann-ON^L algorithm

Note that we use also the dataset AIDS_10K that contains 10 000 graphs. It is a subset that is randomly drawn from AIDS_40K.

6.1.1 Query Sets

For the datasets AIDS_40K, Chem_1M and Syn_10K, we use six query sets Q4, Q8, Q12, Q16, Q20 and Q24. Each set Q_k contains 1 000 query graphs with k edges. For HRPD dataset, we generate only three queries namely, q36, q40, and q44 with size 36, 40, and 44, respectively.

Dataset	$ \mathcal{D} $	Avg. Vertex	Avg. Edge	Dist. Vertex Lab.	Dist. Edge Lab.
AIDS_40K ¹	40 000	25	27	62	3
Chem.1M ²	1 000 000	23.98	25.76	81	3
HRPD	1	9 460	37 000	307	1
Syn_10K	10 000	50	50	3	2

Table 3. Summary statistics of the datasets used in the experiments

6.2 Effects of Optimizations

In this experiment, we show the effect of each optimization independently, and the effect of both of them combined, on the performance of Ullmann-ON^L. For this purpose, we experimented with four versions of Ullmann-ON^L, namely, Ullmann-O that uses only the first optimization Opt1, Ullmann-N¹ that uses only the second optimization Opt2 with $L = 1$, Ullmann-N² that uses only the second optimization Opt2 with $L = 2$, and Ullmann-ON^{1,2} that uses both of the two optimizations and applies the first and the second levels of the neighborhood together in the second optimization Opt2 (i.e. $L = \{1, 2\}$).

Figure 8 plots the results obtained by running the four versions on AIDS_10K (Figure 8 a)), Chem_10K (Figure 8 b)), and Syn_10K (Figure 8 c)) for different query sets. Figure 8 a) shows that Ullmann-N¹ and Ullmann-N² are faster than Ullmann-O except for Q12 and Q16, where Ullmann-O shows the best performance.

In addition to its influence on speed, the first optimization makes the algorithm less sensitive to query size. Ullmann-ON^{1,2} shows the best performance among other versions. It outperforms Ullmann-O, Ullmann-N¹ and Ullmann-N². These results confirm the fact that the two optimizations are neither independent nor conflicting, but they are complementary to each other. Also in Figure 8 b) Ullmann-ON^{1,2} shows the best performance. In Figure 8 c) Ullmann-ON^{1,2} shows the best performance except for Q20 and Q24, where Ullmann-N² is roughly faster than Ullmann-ON^{1,2}.

6.3 Performance Study I

In this section, we compare the performance of the three versions of the proposed algorithm Ullmann-ON¹, Ullmann-ON², and Ullmann-ON^{1,2}.

6.3.1 Search Space Size

Figure 8 also reports the search space size obtained by running the three versions of the proposed algorithm on various datasets (AIDS_40K: Figure 8 d), Chem_1M: Figure 8 e), and Syn_10K: Figure 8 f)). Ullmann-ON^{1,2} always has less search space size compared with Ullmann-ON¹ and Ullmann-ON².

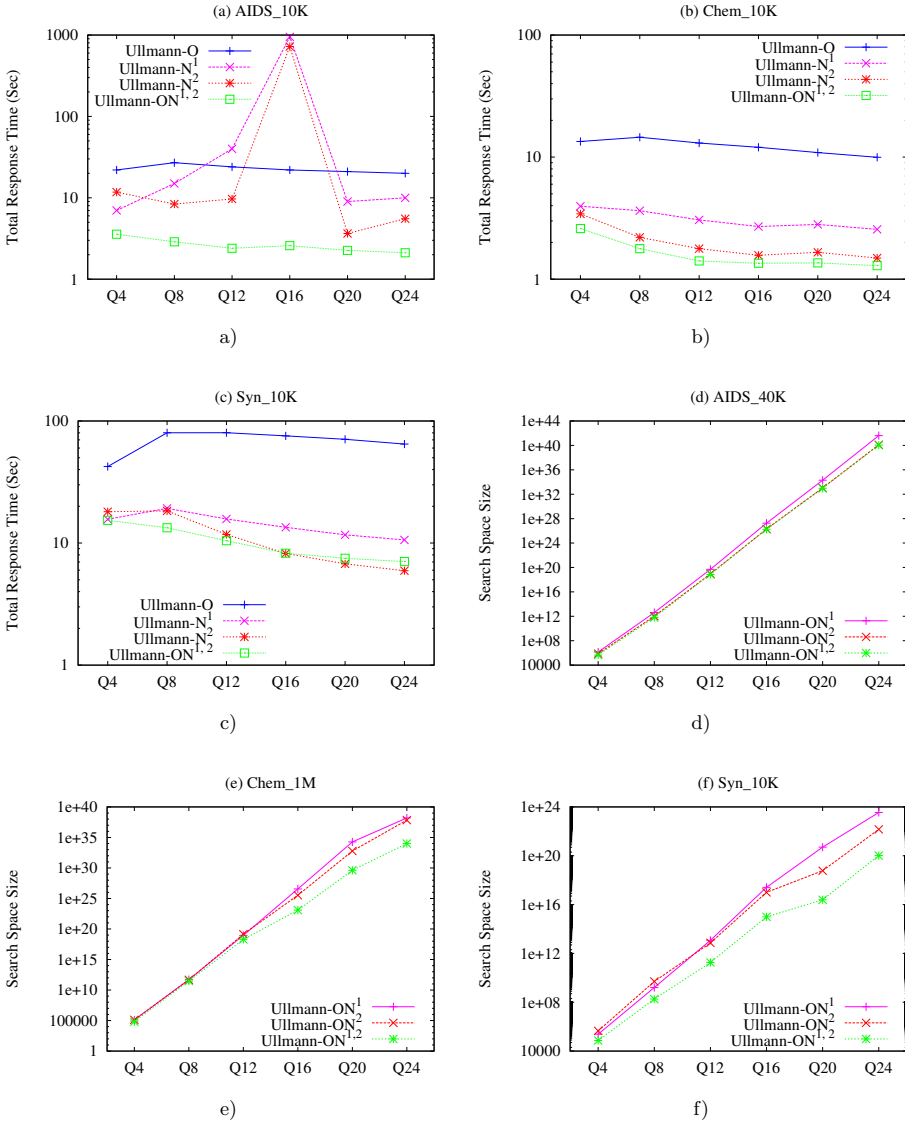


Figure 8. Effects of optimizations [a)-b)-c)] and search space size of the three versions [d)-e)-f)]

6.3.2 Total Response Time: First Embedding and All Embeddings

Figure 9 reports the total response time obtained by running Ullmann-ON¹, Ullmann-ON², and Ullmann-ON^{1,2} on various datasets (AIDS_40K: Figures 9 a), 9 d), Chem_1M: Figures 9 b), 9 e), and Syn_10K: Figures 9 c), 9 f) when we compute the first embedding only and all embeddings, respectively).

We can see that Ullmann-ON² is faster than Ullmann-ON¹ in all figures except Q4 in Figure 9 f) while Ullmann-ON^{1,2} shows the best performance except for Q20 and Q24 in Figures 9 a), 9 c), 9 f).

6.3.3 Scalability: First Embedding and All Embeddings

Figure 10 a) reports the search space size obtained by running the three versions (first embedding) on scalability test of Chem_1M. Here, we used Q8 as query set. The figure shows that the three versions scale linearly and Ullmann-ON^{1,2} has the least search space size. While the Figures 10 b) and 10 c) report the total response time obtained by running the three versions on scalability test of Chem_1M with Q8 when we compute the first embedding only and all embeddings, respectively.

Figure 10 b) shows that Ullmann-ON^{1,2} outperforms Ullmann-ON¹ and Ullmann-ON² by two factor and three factor, respectively. Figure 10 c) shows that Ullmann-ON^{1,2} outperforms Ullmann-ON¹ and Ullmann-ON² by 1.5 factor and more than three factor, respectively.

Recall, in [16], a fair comparison is provided for state-of-the-art subgraph isomorphism algorithms. The overall top three algorithms are vflib3, QuickSI, and DAF. Therefore we compare our method with vflib2 and QuickSI in Section 6.4 and with vflib3 and DAF in Section 6.5.

6.4 Performance Study II

From the performance study I, Ullmann-ON^{1,2} has the best performance among the other versions of the proposed algorithm with respect to time and search space size. Then in this study we compare the performance results of Ullmann-ON^{1,2} algorithm with those obtained on the same datasets by Ullmann³, Vflib2 (abbreviated as Vflib)⁴, and QuickSI (we got its executable from authors).

6.4.1 Total Response Time (Labeled Datasets)

Here, QuickSI was excluded from this experiment since QuickSI works with unlabeled edges datasets only. Figures 10 d), 10 e), 10 f) report the total response time obtained by running Ullmann, Vflib, and Ullmann-ON^{1,2} on various datasets such as AIDS_10K: Figure 10 d), Chem_10K : Figure 10 e), and Syn_10K: Figure 10 f).

³ Ullmann is implemented in C++ with STL and compiled with GNU GCC

⁴ <http://mivia.unisa.it>

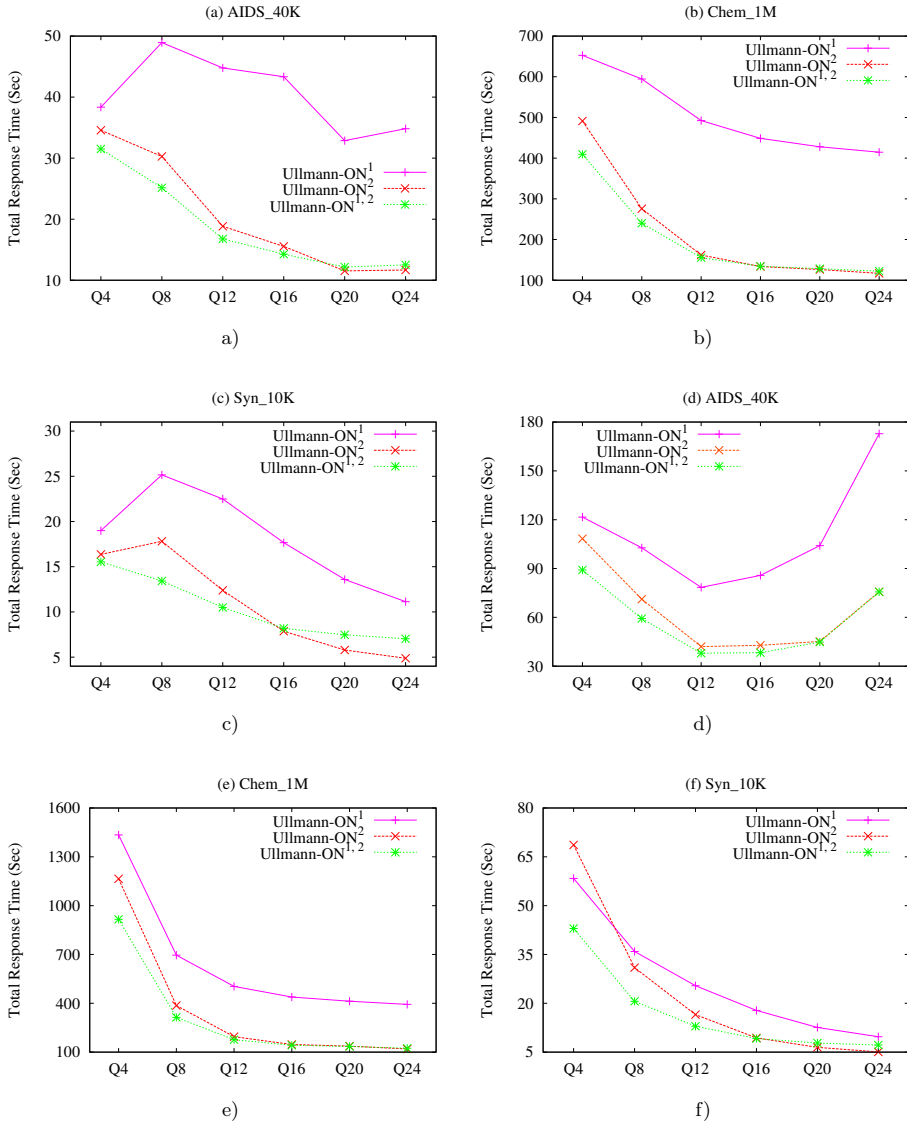


Figure 9. Total response time (first embedding [a)-b)-c]) and all embeddings [d)-e)-f)])

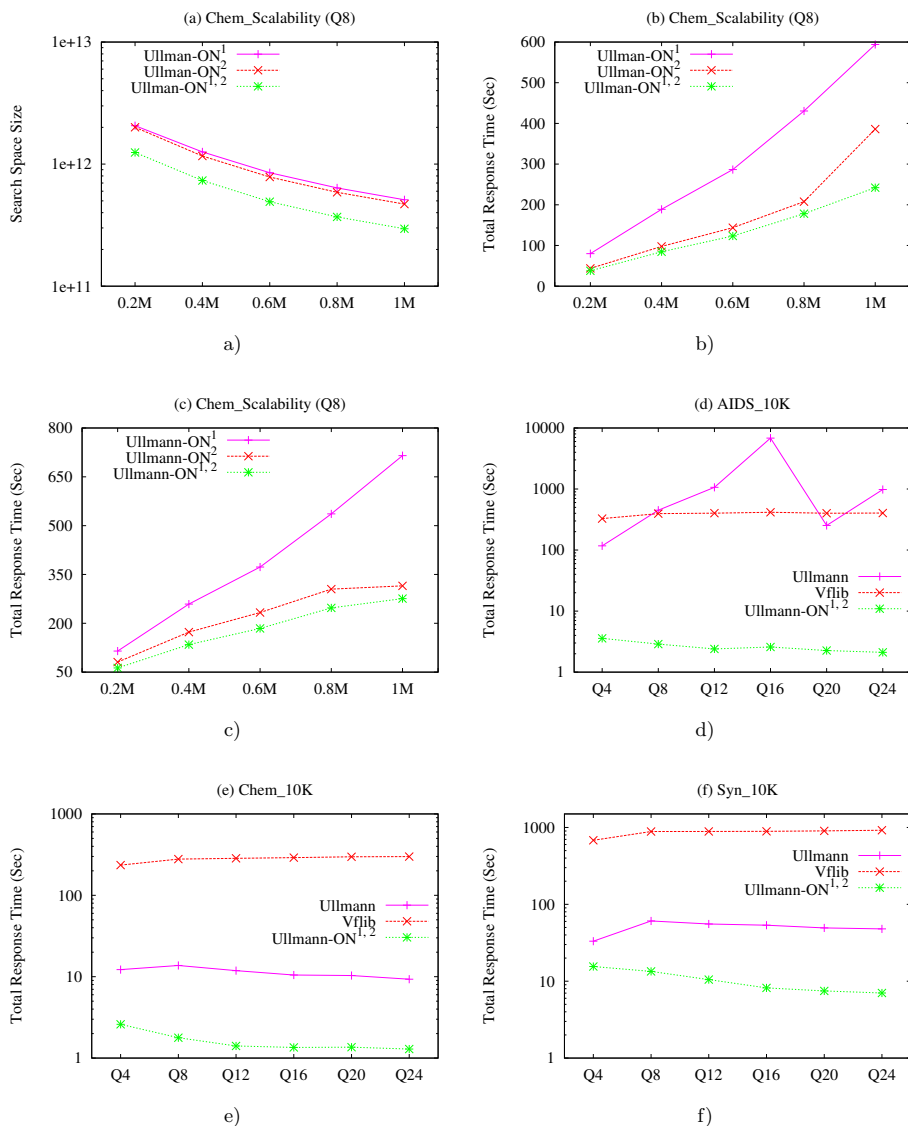


Figure 10. Scalability test (search space size [first embedding: a]), time [first embedding: b]), and time [all embeddings: c]) and total response time [labeled datasets: d)-e)-f])

Ullmann-ON^{1,2} always spends less time compared with Ullmann and Vfib and it is faster by more than two order of magnitude. This happens because Ullmann-ON^{1,2} has better optimizations.

6.4.2 Total Response Time (Unlabeled Datasets)

Here, we used the two datasets AIDS_10K and Chem_10K after removing the edge labels and we denoted them as Unlabeled_AIDS_10K and Unlabeled_Chem_10K. Figures 11 a), 11 b) report the results on the two datasets respectively. From these figures, QuickSI outperforms Ullmann and Vfib on the two datasets. Also, Ullmann-ON^{1,2} shows the best performance, it outperforms Ullmann, Vfib, and QuickSI on Unlabeled_AIDS_10K dataset by more than two order of magnitude, more than one order of magnitude, and more than four factors, respectively (note that Ullman is not shown for the query sets, namely, Q16, Q20, and Q24 since it failed to run on our machine).

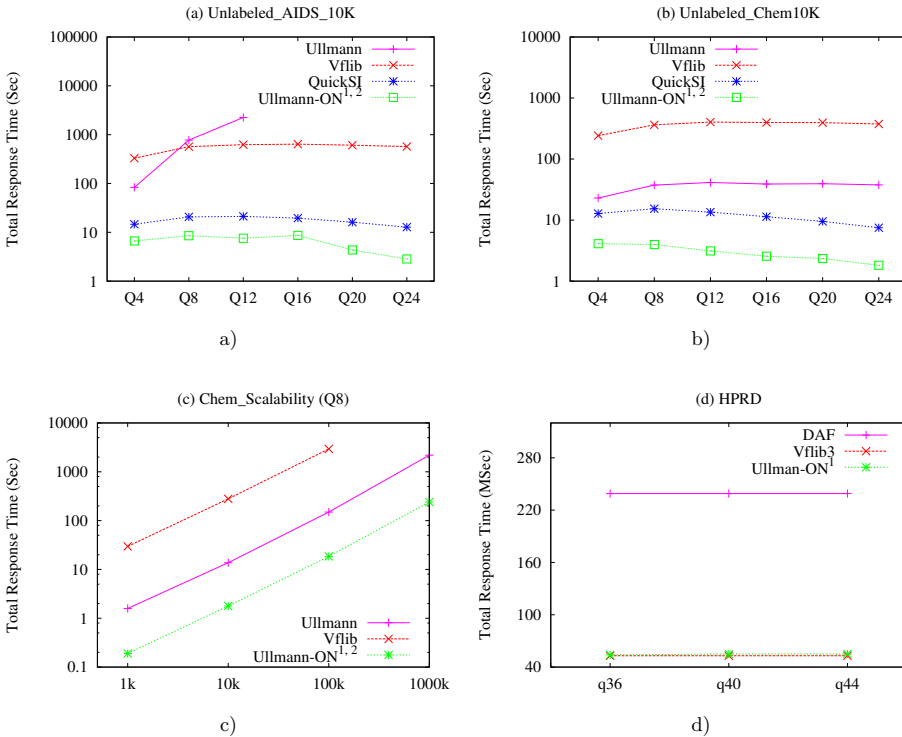


Figure 11. Total response time (unlabeled datasets) [a)-b)], scalability on dataset size [c)], and large dataset [d)]

On Unlabeled_Chem_10K dataset, Ullmann-ON^{1,2} outperforms Ullmann, Vfib, and QuickSI by one order of magnitude, more than two order of magnitude, and more than four factors, respectively.

6.4.3 Scalability

Figure 11 c) shows the scalability of Ullmann, Vfib and Ullmann-ON^{1,2} with respect to the number of graphs using the dataset Chem_1M with Q8. The figure shows that the three algorithms scale linearly. However, Ullmann-ON^{1,2} outperforms Ullmann by three factor, and Vfib by more than two order of magnitude.

Moreover, Vfib is not shown for 1000 K graphs, since it failed to run on large datasets. Recall, QuickSI was excluded from this experiment since QuickSI works with unlabeled edges datasets only.

6.5 Performance Study III

In these experiments, Ullmann, vfib, and QuickSI were excluded due to these algorithms having been originally designed for handling small graphs only. Therefore we compare Ullmann-ON^L against DAF⁵ and Vfib3⁶ on HPRD dataset (one large data graph). Here we set L to 1 in our method. Total response time in MSec for the three queries (q36, q40, and q44) is recorded and demonstrated in Figure 11 d). DAF is the worst in this experiment whereas Ullmann-ON¹ and Vfib3 have approximately the same performance.

7 CONCLUSION

In this paper, we proposed an efficient algorithm for *SGI* called Ullmann-ON^L. Ullmann-ON^L minimizes sharply the search space by sorting the query vertices in efficient way and by using the different levels of vertex neighborhoods. We implemented three versions of the proposed algorithm (Ullmann-ON¹, Ullmann-ON², and Ullmann-ON^{1,2}) and we found that Ullmann-ON^{1,2} version has the best performance with respect to time and search space size on small data graphs. Experimental results on real and synthetic datasets demonstrate that Ullmann-ON^{1,2} outperforms Ullmann and Vfib by 1-3 order of magnitude and outperforms QuickSI verifier by more than four factors. Ullmann-ON^{1,2} has excellent scale-up properties with respect to the number of graphs. In large dataset, Ullmann-ON¹ has the best performance among the other versions and it outperforms DAF by more than four factors and has the same performance as in Vfib3.

⁵ <http://github.com/SNUCSE-CTA/DAF>

⁶ <http://mivia.unisa.it>

REFERENCES

- [1] PETRAKIS, E.—FALOUTSOS, A.: Similarity Searching in Medical Image Databases. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, 1997, No. 3, pp. 435–447, doi: 10.1109/69.599932.
- [2] CAI, D.—SHAO, Z.—HE, X.—YAN, X.—HAN, J.: Community Mining from Multi-Relational Networks. In: Jorge, A. M., Torgo, L., Brazdil, P., Camacho, R., Gama, J. (Eds.): *Knowledge Discovery in Databases: PKDD 2005*. Springer, Berlin, Heidelberg, *Lecture Notes in Computer Science*, Vol. 3721, 2005, pp. 445–452, doi: 10.1007/11564126.44.
- [3] WILLETT, P.—BARNARD, J. M.—DOWNS, G. M.: Chemical Similarity Searching. *Journal of Chemical Information and Computer Sciences*, Vol. 38, 1998, No. 6, pp. 983–996, doi: 10.1021/ci9800211.
- [4] GAREY, M. R.—JOHNSON, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.
- [5] GOUDA, K.—HASSAAN, M.: A Fast Algorithm for Subgraph Search Problem. 2012 8th International Conference on Informatics and Systems (INFOS), 2012, pp. DE-53–DE-59.
- [6] ULLMANN, J. R.: An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, Vol. 23, 1976, No. 1, pp. 31–42, doi: 10.1145/321921.321925.
- [7] CORDELLA, L.—FOGGIA, P.—SANSONE, C.—VENTO, M.: A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 26, 2004, No. 10, pp. 1367–1372, doi: 10.1109/TPAMI.2004.75.
- [8] SHANG, H.—ZHANG, Y.—LIN, X.—YU, J. X.: Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *Proceeding of the VLDB Endowment*, Vol. 1, 2008, No. 1, pp. 364–375, doi: 10.14778/1453856.1453899.
- [9] LAND, A. H.—DOIG, A. G.: An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, Vol. 28, 1960, No. 3, pp. 497–520, doi: 10.2307/1910129.
- [10] CARLETTI, V.—FOGGIA, P.—SAGGESE, A.—VENTO, M.: Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 40, 2018, No. 4, pp. 804–818, doi: 10.1109/TPAMI.2017.2696940.
- [11] ZHAO, P.—HAN, J.: On Graph Query Optimization in Large Networks. *Proceeding of the VLDB Endowment*, Vol. 3, 2010, No. 1-2, pp. 340–351, doi: 10.14778/1920841.1920887.
- [12] HAN, W. S.—LEE, J.—LEE, J. H.: Turboiso: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*, 2013, pp. 337–348, doi: 10.1145/2463676.2465300.
- [13] ZHANG, S.—LI, S.—YANG, J.: GADDI: Distance Index Based Subgraph Matching in Biological Networks. *Proceedings of the 12th International Conference on Ex-*

- tending Database Technology: Advances in Database Technology (EDBT '09), ACM, 2009, pp. 192–203, doi: 10.1145/1516360.1516384.
- [14] REN, X.—WANG, J.: Exploiting Vertex Relationships in Speeding Up Subgraph Isomorphism over Large Graphs. Proceeding of the VLDB Endowment, Vol. 8, 2015, No. 5, pp. 617–628, doi: 10.14778/2735479.2735493.
- [15] HAN, M.—KIM, H.—GU, G.—PARK, K.—HAN, W. S.: Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19), ACM, 2019, pp. 1429–1446, doi: 10.1145/3299869.3319880.
- [16] ZENG, L.—JIANG, Y.—LU, W.—ZOU, L.: Deep Analysis on Subgraph Isomorphism. 2020, doi: 10.48550/arXiv.2012.06802.



Karam GOUDA is Dean of the Faculty of Computers and Artificial Intelligence, Benha University, Egypt. His research interests are string data management, data mining, and query processing in graph databases.



Gyöngyi BUJDOSÓ is Senior Lecturer at the Faculty of Informatics, University of Debrecen, Debrecen, Hungary. Her research interests are cybersecurity and privacy awareness, web design, disruptive technologies in education, and integration of on-line visual communications into on-line education.



Mosab HASSAAN is Lecturer of computer science at the Department of Mathematics and Computer Science, Faculty of Science, Benha University, Egypt. His research interests are query processing in graph databases and data mining tasks.