Labeled Subgraph Matching Using Degree Filtering

Lixin Fu and Surya Prakash R Kommireddy
Department of Computer Science,
University of North Carolina at Greensboro,
167 Petty Building, Greensboro, NC 27412, USA.
phone: 336-256-1137; fax: 336-256-0439; e-mail: lfu@uncg.edu

Abstract - *Subgraph isomorphism (SGI) is the problem to determine whether there is a subgraph in the given larger graph (called model graph) that is identical to a given smaller graph (called query graph). It is well known that SGI problem for an unlabeled graph in general is NP Complete. The famous Ullman's algorithm is still used in popular subgraph matching software package such as POSSUM. However, this algorithm handles unlabeled graphs. In this paper, we design and implement a new SGI algorithm that can handle graphs with labeled edges. Such graphs have important applications e.g. cheminformatics and pattern recognition. Our major contribution is to integrate degree filtering while comparing the node labels, so that the performance is greatly improved.*

Keywords: graph algorithms, subgraph isomorphism, labeled graphs, matching

1 Introduction

Many real world problems can be represented in term of graphs. From late seventies, graph-based techniques have been proposed as a powerful tool for pattern recognition. Pattern recognition in chemistry or biological databases is modeled as graph matching problem. Graph matching problems are of two types, they are exact graph matching and approximate graph matching. In this paper, we mainly concentrate on sub-graph isomorphism which is in category of exact graph matching.

The main use of the subgraph isomorphism in chem-Informatics is "the chemical similarity between any two molecules, either at the sub or superstructure level, and clustering of similar molecules are widely used to measure the diversity of chemical space and these methods are important as they can be applied towards discovering any new drug like molecules" In addition to cheminformatics, SGI has many other applications in bioinformatics, scene analysis, pattern recognition , image processing, etc. Related work is given in the next section. In sec. 3, Ullman's algorithm is discussed as the base for our algorithm (in sec. 4) to compare with. The experiment section of 5 shows that our new algorithm outperforms the traditional Ullman's algorithm. Lastly, the conclusion and further work are given.

.

2 Related Work

2.1 Exact Matching Algorithms

Subgraph isomorphism will come under the exact matching algorithms category. Exact graph matching is requires the mappings between the nodes of the two graphs to be edge-preserving. That is, if any two nodes in the first graph are linked by an edge then they are mapped to two nodes in the second graph that are linked by an edge as well. In the case of labeled graphs, the corresponding node and edge labels must match as well. There is a bi-directional one-to-one correspondence between each node of the first graph to the each node of subgraph of the second graph.

Most of the algorithms for exact graph matching are based on some form of tree search with backtracking. Many different implementations have been employed. Among them the recursive depth first search uses less memory.

Ullman‟s algorithm was published as early as 1976, which is is still widely used today in the famous software package POSSUM (Protein On-line Substructure Searching – Ullman Method) [4]. Ullman proposes so-called refinement procedure that works on a matrix of possible future matched node pairs to remove. Many algorithms are proposed based on this algorithm.

Another similar algorithm is Corneil's breadth-first algorithm, which is presently the core component of Gemini and Sub-Gemini, which is still best performing package today for sub circuits extraction in VLSI layout verification. [2].
Cordella proposed a new algorithm for the subgraph isomorphism called VF [5] algorithm. In this algorithm, a heuristic is based on the analysis of the sets of nodes adjacent to the ones already considered in the partial mapping. The author redesigned and proposes another algorithm called VF2 [6] which is significant improvement over Ullman‟s and work well for large graphs also.

Subgraph isomorphism has been proposed by the Larrosa and Valiente [7], authors changed slightly the problem and stated as constraint satisfaction problem (CSP), and this problem has been helpful in the framework of discrete optimization and operational research.

In this paper, we mainly modified Ullman's method so that in the case of labeled graph, the performance is greatly improved due to label matching and degree filtering. We will

first present the simple enumeration algorithm and then the idea of Ullmann's method, and in coming section our algorithm called SGI-DF (Subgraph Isomorphism with Degree Filtering).

2.2 Simple Enumeration Algorithm for Finding Subgraph Isomorphism

This algorithm describes a brute-force enumeration procedure that is actually a depth-first tree-search algorithm. This algorithm is designed to find all the isomorphisms between given graphs G1 and the subgraphs of G2. The adjacency matrices for graphs G1 and G2 are A= [αij] and B= [βij].

These methods will use the representation of adjacency matrices. In this algorithm, we introduce the very important concept of permutation matrix, which is a key concept in Ullman's algorithm. We call the permutation matrix as M'. of size (rows of matrix A) X (rows of matrix B), such that the permutation matrix whose elements are 1's and 0's and such that each row contains exactly one 1 and no column contains more than one 1.

The permutation matrix is used to find the matrix C, where C= [cij] = M'(M'B)^T, and T represents transpose. Subgraph isomorphism exists if for all i and j's of α and β, (α_{ij}=1) = (c_{ij}=1).

2.3 Permutation Matrix

A permutation matrix, of N×M, has exactly one entry 1 in each row and each column and 0s elsewhere. For a 4×4 permutation matrix, 4! (Factorial of 4) permutation matrices are possible.

In the simple brute force enumeration procedure, it needs all the permutation matrices to be tried until the subgraph isomorphism exists. Here are just 3 of the matrices in all 24 matrices.

1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1	0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

Example is given below for one of the permutation matrices [3 2 1 0], how it will be computed

3 2 1 0
3r column and 0th row- 1
2nd column and 1st row-1
1st column and 2rd row-1
0th column and 3rd row-1
(remaining all are 0's)

	0	**1**	**2**	**3**
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

Table 1: permutation matrix showing each row and column

3 Ullman's Algorithm

To reduce the amount of computation, Ullmann proposed a refinement procedure to eliminate some of the 1's from the matrices M, thus reducing the number of the possibility matrices.

Ullman's algorithm mainly depends on permutation matrices to find all the isomorphism for given two graphs. Using permutation matrices, we find the matrix C and then compare it with a query graph. Ullmann's method is based on backtracking and a refinement procedure.

The inputs are the model graph and query graph, and the output will be a permutation matrix.

Query graph G= (V, E, L_V, L_E)

Model graph G1= (V1, E1, L_V, L_E)

Ullmann's algorithm:

ULLMANN (G= (V, E, L_V, L_E), G1= (V1, E1, L_V, L_E))

1. Let P = (p_{ij}) be a n×n permutation matrix, n = |V|,
   m = |V1|,and M=adjacency matrix of G,
   M1=adjacency matrix of G1

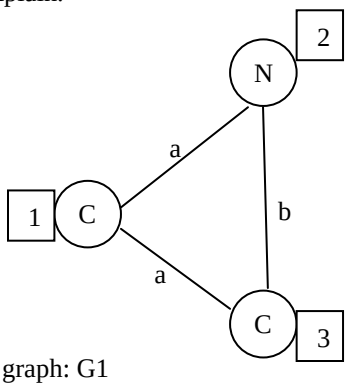
2. Call Backtrack(M; M1;P;1)

3. Procedure Backtrack (adjacency matrix M, adjacency matrix M1, Permutation matrix P, counter k)
   (a) if k > m then P represents a subgraph isomorphism from GI to G. Output P and return.
   (b) For all i = 1 to n
      i. set P_{ki} = 1 and for all j = i set P_{kj} = 0
      ii. if $S_{k,k}$(M1) = $S_{k,n}$(P) M ($S_{k,n}$(P))T then
         call Backtrack(M, M1, P, k +1)

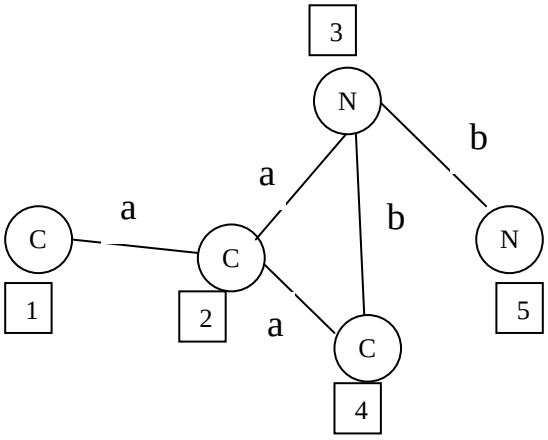
Ullmann followed the backtracking procedure with the refinement steps. The refinement procedure is done by the idea of forward checking. The backtrack procedure in Ullmann's algorithm recursively builds permutation matrix element by element, checking for a match with input graph at each step.

The backtracking procedure stops whenever the match found or until all the possible matrices are over.

In comparison with the simple enumeration procedure, Ullmann's refinement method will reduce the number of permutation matrices. We will use the following example to explain.



graph: G1



graph: G2

In the above figure, circles represent the node labels, squares represent indexes and remaining are the edge labels.
For the simple enumeration procedure, the permutation matrix would contain of all 1's.

Indexes(labels)	1(C)	2(C)	3(N)	4(C)	5(N)
1(C)	1	1	1	1	1
2(N)	1	1	1	1	1
3(C)	1	1	1	1	1

In the above matrix, the total combinations would be 5*4*3=60 possibilities of permutation matrices. Because of its brute force approach it should have to try every possibility in worst case. For Ullmann's procedure, the permutation matrix would contain fewer 1's compared to the simple enumeration procedure.

Indexes(labels)	1(C)	2(C)	3(N)	4(C)	5(N)
1(C)	1	1	0	1	0
2(N)	0	0	1	0	1
3(C)	1	1	0	1	0

In this case, there are only 12 different possibilities in worst case.

4 Label Matching with Degree Filtering

To further improve Ullmann's method, we have to decrease the number of 1's in the permutation matrix, so that we get smaller number of combinations to check for the subgraph isomorphism. Our idea is to add degree filtering and label matching features for labeled graphs.

Using our algorithm we can reduce a lot of 1's in the permutation matrix than the 1's in the Ullmann's algorithm, thus reducing many unnecessary comparisons from query graph to the main graph.

The degree of a node is the total number of neighboring nodes connected to the node. For example, in Graph G2, the degrees of the nodes 1 through 5 are 1, 3, 3, 2, and 1 respectively.

4.1 Design of the algorithm

As we mentioned before, the aim of algorithm is to reduce the non-zero entries of matrix M to zeros. Suppose α_1,α_2,……..α_n are in query graph and β_1, β_2,……. B_m are in model graph. A and B are adjacency matrices of query and model graphs.

Step 1: Fast elimination

Before this step, we will find degree for each node in query graph and input graphs.
In this step, we will check the maximum degree of query graph and compare to that of the model graph, if there is no node having same label and not having higher degree than the maximum degree of query graph, then it's going to be terminated and say there is no subgraph form G1 to G2.

Step 2: Preprocessing

In this step, we preprocess the matrix and reduce as many entries of M to zero as possible by allowing only vertices with the same or greater degree to be mapped to each other as well as checking the label names.

We construct mapping matrix M as follows.
M(i,j) = 1, if Degree(α_i) ≤ Degree(β_j) and Label(α_i)=Label(β_j)
M(i,j) = 0, Otherwise

The above 2 steps are not in the Ullmann's method and so we improve the performance, and the remaining steps are very much similar.

Step 3: Changing Mapping matrix to the different arrays, and then using that arrays we will generate distinct arrays, each array for each possible matrix.

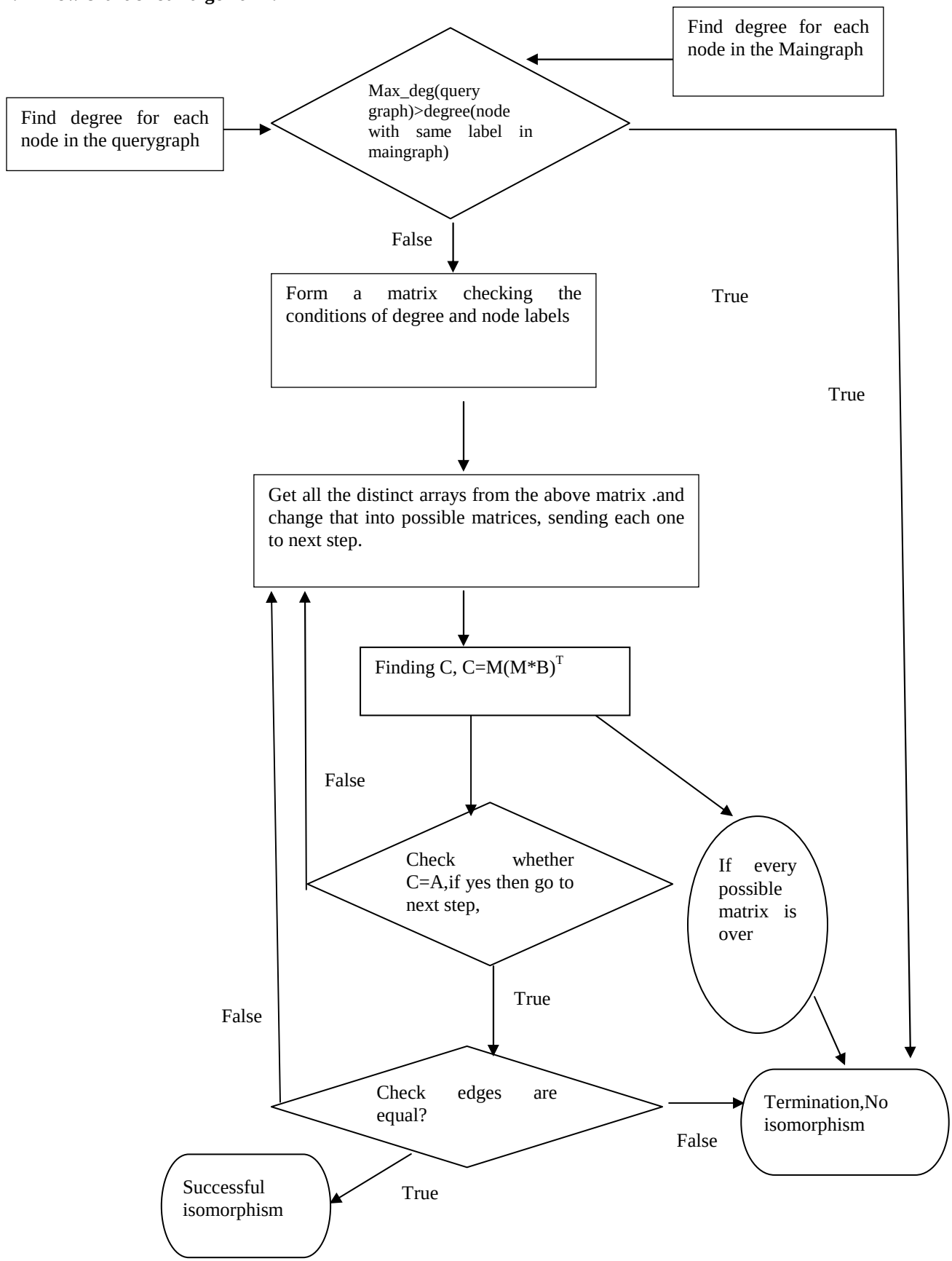
Step 4: Computing C according to this formula,

C=M(M*B)T, where M is the each possible matrix,
B is the input matrix
T is the transpose
C is the final matrix to check each time to the query matrix for every possible matrix.

Step 5:

If C=A, then the possible matrix matches the structure of query and main graphs, then it will check all the edges in query and main graph ,if they are same then print the result, If not then go to step 3 and get other possible matrices.

4.2 Flow chart of our algorithm:



In the example of section III, we know that there are 12 possible combination matrices in Ullmann's algorithm.
Now for our algorithm:
First find degrees for query and main graph.
For query graph:
Deg(1)=2
Deg(2)=2
Deg(3)=2
For the main graph:
Deg(1)=1
Deg(2)=3
Deg(3)=3
Deg(4)=2
Deg(5)=1

Now, Max_degree(query graph)=2 < Deg(2)
The entries are in the form of: Indexes (labels) [Degree]

	1(C)[1]	2(C)[3]	3(N)[3]	4(C)[2]	5(N)[1]
1(C)[2]	0	1	0	1	0
2(N)[2]	0	0	1	0	0
3(C)[2]	0	1	0	1	0

Now the combinations in above matrices without repetition are 2, much smaller than Ullman's 12. If we take large graphs, then we can clearly see that using degree filtering and label matching may greatly improve the performance.

5 Experiments

We used the data sets in a standard graph library from (http://www.cs.ucsb.edu/~xyan/software.htm/). The graph library consists of datasets with the query and model graphs. We used java on a windows 7 machine for the whole implementation of Ullmann's algorithm as well as our algorithm. We tested our algorithm on datasets taken in the form of adjacency lists. When the query size grows the running time is also increasing as well. We tested the same datasets for the Ullmann's algorithm as well.
We compare the runtimes of our algorithm and those of Ullmanns algorithm when the query size and number of nodes in the main graphs increase.

Results:

In all the diagrams, x-axis represents the number of model graph nodes and y-axis is the runtime in mili-seconds. From these figures, we can see that our algorithm beats Ullman's algorithm for different query sizes. Furthermore, the larger the sizes, the more number of nodes, our performance advantage is also larger.

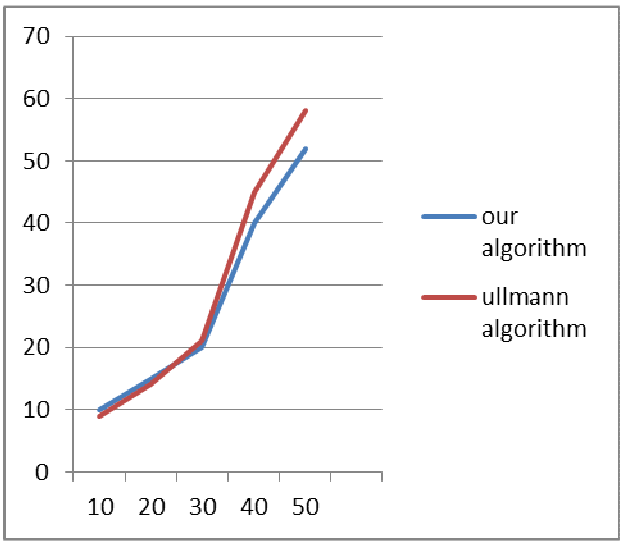


Figure 1: for query size of 3.

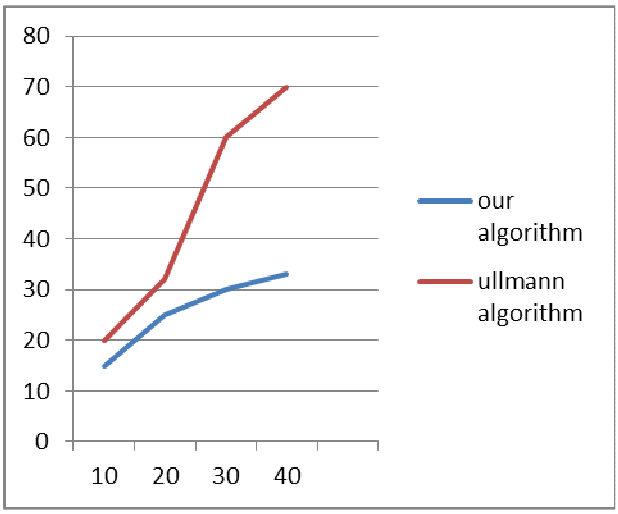


Figure 2: for query size of 5.

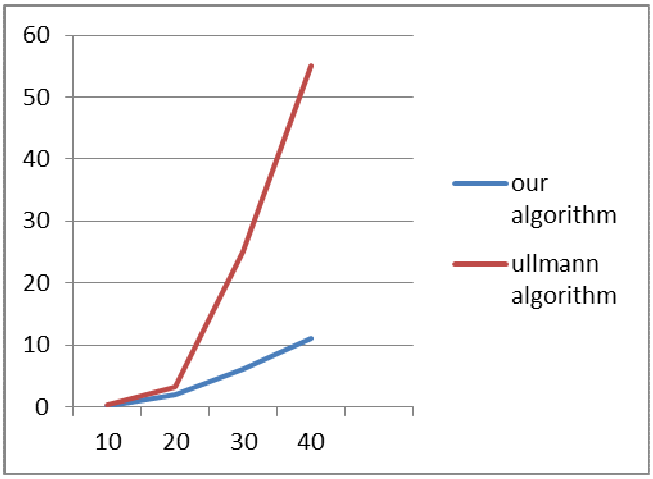


Figure 3: for Query size 7.

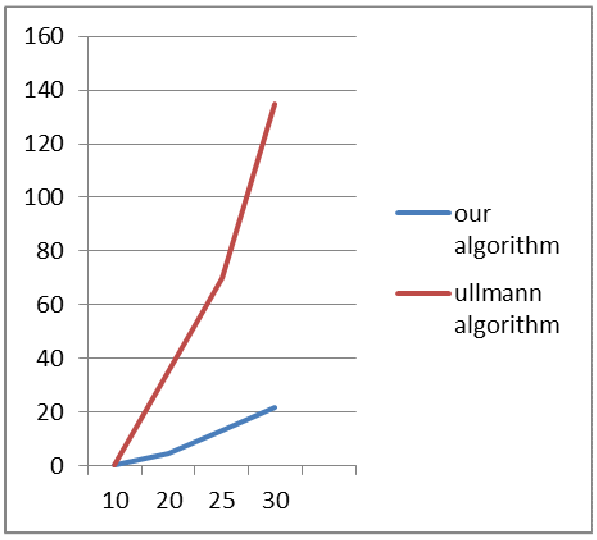


Figure 4: for query size 9.

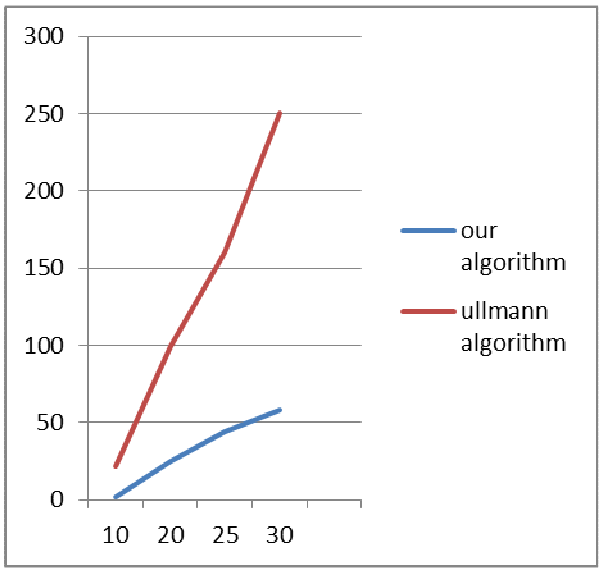


Figure 5: for query size 11.

6 Conclusion

We proposed a new algorithm called SGI-DF, which works for labeled graph subgraph isomorphism. Our conclusion from the experiments and results is that when the query size is smaller there would be no great difference but when the query size becomes bigger the runtime difference between Ullman's algorithm and SGI-DF becomes much larger. The reason is that our degree filtering feature added can eliminate many unnecessary computations. When the query sizes grow both algorithms run longer. In addition, our algorithm works for the labeled edges also which is very helpful in applications such as chem-informatics.

The scalability of this algorithm remains to be verified due to limited computing resources. One area is to extend the work here to accommodate large graphs stored in external disks. Further work on SGI computing on cloud environment is also desired.

References

[1] Taming verification hardness: an efficient algorithm for testing subgraph isomorphism, H Shang, Y Zhang, X Lin – proceedings of the VLDB Endowment, (2008)

[2] An Effective Approach for Solving Subgraph Isomorphism Problem- Zong Ling, Department of Electrical Engineering, University of Hawaii (1996)

[3] Parallel Subgraph Isomorphism, Aaron Blankstein, Matthew Goldstein, MIT Computer Science and Artificial Intelligence Laboratory (2010)

[4] J. R. Ullman, An algorithm for subgraph isomorphism, Journal of the association of computing machinery, 23(1976) 31-42.

[5] Performance evaluation of VF graph matching algorithm, LP Cordella, P Foggia, C Sansone, Image Analysis and processing, (1999) pp.1172-1177.

[6] A (Sub)Graph Isomorphism Algorithm for Matching large graphs- Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento, Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition, (2001), pp. 149-159.

[7] Liu and D. J. Klein. The graph isomorphism problem. Journal of Computational Chemistry, 12(10): 1243-1251, 1991.

[8]http://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

[9] Subgraph Isomorphism in Polynomial Time – B.T. Messmer and H. Bunke, University of Bern, Neubruckstr. 10, Bern, Switzerland, Recent Developments in Computer Vision, (1996).