An Efficient Computation of Reachability Labeling for Graph Pattern Matching

Bhargavi Balla, Supreethi Kalyandurgam Pujari
Department of Computer Science & Engineering
JNTUH College of Engineering
Hyderabad, India
Emails: {bhargavi.bbv1, supreethi.pujari}@gmail.com

Abstract— **Due to the rapid growth of Internet, most of the data that is available in the Internet that is archived/ analyzed, is graph structured in nature as graphs form a powerful modeling tool. The problem of graph pattern matching is to find all the tuples that match a user-given graph pattern from a large directed graph. For faster access of paths in the large directed graph, transitive closure of the graph is compressed and maintained using 2-hop reachability labeling technique by assigning every node a 2-hop label. These 2-hop labels are computed using a geometry-based approach that will be useful in solving the graph pattern matching problem. In this paper, a geometry-based approach that computes the 2-hop reachability labeling is described. The experimental results show that the proposed approach efficiently computes the compressed transitive closure technique of reachability labeling.**

Keywords- **graph pattern; graph matching; 2-hop cluster; 2-hop labeling; 2-hop cover.**

I. INTRODUCTION

Graphs form a powerful modeling tool to represent various networks in different areas like chemistry, biology, web, etc. In online social networking systems like Facebook and Twitter, the relationships among users and their proximity can be conveniently expressed using graphs. Thus, there is a demand for efficiently querying the graph data.

Graph database [1] is a large labeled directed graph or a collection of labeled directed graphs. A graph pattern is a sequence of nodes and edges which is constructed by connecting nodes based on links/relationships between them as required by the user. Given a graph database and a graph pattern, finding all the set of tuples (an ordered sequence of vertices) that match a user given graph pattern is the graph pattern matching problem. For instance, in analyzing online social networking systems, a large graph can be obtained where the job-title attribute on each node can be regarded as label. A small graph pattern can be to discover connections between several people with specified jobs. But, the graph pattern matching problem is challenging as graph data can be large and graph patterns can be large and complex.

To access the paths in a large graph data faster, its compressed transitive closure is pre-computed using 2-hop reachability technique which involves assigning a graph code termed 2-hop reachability label to each node of the directed graph. The computation of 2-hop reachability labeling for the graph is found to be NP-hard [3]. In this paper, a geometry-based approach is implemented to efficiently compute the 2-hop reachability labels for a large directed graph which is a nearly optimal solution. The graph codes computed will be useful in solving the graph pattern matching problem in relational database context [1].

Section II covers the related work done for finding the efficient techniques to solve the problem of graph pattern matching and 2-hop reachability labeling. Section III describes the prominent compressed transitive closure techniques while section IV describes the procedure to compute 2-hop reachability labels efficiently. Section V reveals the experimental results and analysis and in section VI, we conclude the paper with future work.

II. RELATED WORK

Extensive survey has been done for finding efficient techniques to solve the graph pattern matching problem [10]. It includes the survey on tree-pattern matching techniques [8][9], graph pattern matching techniques [1][5] and extensive survey on multi-interval encoding [4] and 2-hop labeling [2][3][6][7].The problem of tree pattern matching is to find the set of patterns from a large tree that match the given tree pattern. Bruno et al. [8] used stack encoding scheme for tree pattern matching in XML documents with elements and parent-child relationships rendering it as tree. Chen et al. [9] further improved by using hierarchical stack encoding scheme for tree pattern matching. But, these techniques do not work on graph data directly as graphs do not have the good acyclic property of trees.

For faster access of paths and for testing if two nodes are reachable, transitive closure is pre-computed and stored in compressed form. Multi-interval encoding defined by Agrawal et al. [4] is a compressed transitive closure technique used for faster processing of graph pattern matching of graph-structured documents in [5]. The 2-hop reachability labeling defined by Cohen et al. [3] is a compressed transitive closure technique where each node is assigned labels that represent the reachability information of the node. The problem of 2-hop cover is to find the minimum size of 2-hop cover for a given graph, which is proved to be NP-hard [3]. Cohen et al. [3] show that a greedy algorithm exists to compute a nearly optimal solution for the 2-hop cover problem. The resulting size of

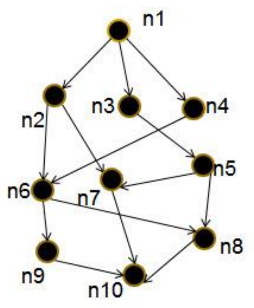

Figure 1. A Directed Graph G

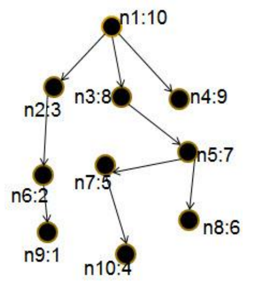

Figure 2. Optimum tree cover with postid for each node

the greedy algorithm is larger than the optimal by atmost O(log n). The basic idea is to solve the minimum 2-hop cover problem as a minimum set cover problem [3]. Schenkel et al. [6][7] implemented divide and conquer approach where the directed graph is partitioned into subgraphs and 2-hop cover is computed for each partition as in [3] and then the results are combined. The approach developed by Cohen et al. focused on finding minimum overlap among the subsets for finding 2-hops of the graph while J. Cheng et al. [2] implemented a geometry-based approach and focused on finding the minimum number of subsets which finds the 2-hops faster than the former approach.

The survey [10] resulted in discovering an efficient geometry-based approach [2] for finding reachability labels of the directed graph.

III. COMPRESSED TRANSITIVE CLOSURE COMPUTATION TECHNIQUES

Transitive closure represents all the set of paths between the nodes of the graph that satisfy transitivity property. The transitive closure size is defined as the total number of paths present in the transitive closure. By pre-computing transitive closure, we can access shortest paths faster and check the existence of paths between the two nodes. But, the transitive closure size is very large compared to the total number of vertices/edges that represent the directed graph. The following are the two different techniques to compute the compressed transitive closure efficiently.

A. *Multi-interval Encoding*

Multi-interval encoding technique involves assigning every node a postid and interval list with atleast one interval that together represent the compressed transitive closure of the directed graph.

Initially, the directed graph is converted into DAG (directed acyclic graph) by computing the maximal strongly connected components of the directed graph and assigning a node to represent each maximal strongly connected component in DAG. Then, multi-intervals are computed using Agrawal et al. algorithm [4]. To compute multi-intervals, an optimum tree cover is derived first from the DAG [4]. Then, postids (the numbers) are assigned to the nodes in post-order traversal order of the optimum tree.

For instance, consider the optimum tree cover shown in Fig. 2 of the directed graph G in Fig. 1. In Fig. 2, consider a node “n9” to which the assigned postid is 1, which is assigned based on the post-order traversal of the optimum tree. To each node of the optimum tree cover, an interval [s, e] is assigned where ’e’ is the postid of the current node and‘s’ is the postid of the lowest descendant node.

For each leaf node ‘v’ with postid ‘i’, its interval assigned is [i, i] and that of its parent node with postid ‘j’ is [i, j]. For instance, from Fig. 2, the interval of the leaf node “n9” is [1, 1] and interval of its parent node “n6” is [1, 2] (shown in Table I). Multiple intervals for each node of the graph come into existence if there are back edges to the nodes in the DAG. For instance, in Fig. 1, there is a back edge (n8, n10) for node “n8” hence the interval of node “n10”, i.e. [4, 4] is added to the interval list of “n8” as shown in Table I.

In general, there exists a path a~b if and only if the postid of ‘b’ lies in atleast one interval in the interval list of ‘a’. For instance, there exists a path n3~n8 as the postid of “n8” is 6 which lies in the interval [4, 8] of n3. Multi-intervals for the other nodes of the directed graph are same

TABLE I. MULTI-INTERVAL ENCODING OF G

V	pid	I
n1	10	[1, 10]
n2	3	[1, 6]
n3	8	[4, 8]
n4	9	[[1, 2], [4, 4], [6, 6], [9, 9]]
n5	7	[4, 7]
n6	2	[[1, 2], [4, 4], [6, 6]]
n7	5	[4, 5]
n8	6	[[4, 4], [6, 6]]
n9	1	[[1, 1], [4, 4]]
n10	4	[4, 4]

as the multi-intervals of the maximal strongly connected nodes of the directed graph that represent them in DAG. Table I shows the multi-interval encoding of each node of G shown in Fig. 1. Thus, multi-interval encoding represents the compressed transitive closure of the directed graph.

The disadvantage of multi-interval encoding is that, it is lengthy and its storage cost increases with increase in the number of vertices/edges of the directed graph and sorting is required to perform graph pattern matching using the multi-interval encoding technique. Hence, the more efficient 2-hop reachability labeling or 2-hop cover technique is opted to compute and store the compressed transitive closure.

B. 2-hop Reachability Labelling

A hop in a directed graph is defined by a path in the graph and one of the end points of the path. For each node 'v' in a directed graph G (V, E), a label $L(v) = \{L_{in}(v), L_{out}(v)\}$ is assigned where $L_{in}(v)$ represents the set of the nodes in G that can reach 'v' & $L_{out}(v)$ represents the set of nodes in G that are reachable from 'v', (hence the name 2-hop) which define the 2-hop reachability labeling [3].

A 2–hop cover is a 2–hop labeling of directed graph G such that if there is a path u~v in G, then $L_{out}(u) \cap L_{in}(v) \neq \emptyset$. 2-hop cover is computed such that the transitive closure of graph is covered.

Table II shows the 2-hop reachability labels for the nodes of the directed graph G of Fig. 1. For instance, L (n3) = {{n1}, {n5}} is the 2-hop reachability label for a node 'n3' where L_{in} (n3) = {n1} & L_{out} (n3) = {n5}. There exists a path n3~n8 as L_{out} (n3) ∩ L_{in} (n8) is {n5}. Thus, 2-hop reachability labelling represents the compressed transitive closure of the directed graph.

The problem of finding 2-hop cover is to assign the 2-hop reachability labels such that the total size is minimum which is found to be an NP-hard problem as it can be reducible to minimum set-cover problem which has no optimal solution. Minimum set cover problem is to find the subsets with minimum overlap covering all the paths. Each subset has a center w associated with it represented as $S(F_w, w, T_w)$ which is termed the 2-hop cluster with center 'w' and

TABLE II. 2-HOP REACHABILITY LABELING OF G

V	Lin(v)	Lout(v)
n1	{}	{n5, n6}
n2	{n1}	{n6}
n3	{n1}	{n5}
n4	{n1}	{n6}
n5	{}	{n1, n3}
n6	{}	{}
n7	{n2, n5}	{n10}
n8	{n5, n6}	{n10}
n9	{n6}	{n10}
n10	{n5, n6}	{}

F_w and T_w constitute all the set of nodes that are reachable from 'w' and that can reach 'w' respectively. The center with maximum cost is selected. The cost is assigned to the center node based on the criterion of maximum number of paths that the node can cover.

IV. EFFICIENT COMPUTATION OF 2-HOP LABELLING FROM MULTI-INTERVAL ENCODING USING GEOMETRY-BASED APPROACH

We computed 2-hop clusters by implementing the geometry-based concept behind the algorithm in [2]. The geometry-based technique involves the following steps.

1. Construction of virtual reachability map.
2. Computation of rectangular map for each node 'w' that forms a bipartite graph with center 'w'.
3. Derivation of 2-hop cluster by mapping the rectangular map that has maximum matches with the virtual reachability map.

The steps 2 and 3 are repeated until the virtual reachability map is completely covered. The set of 2-hop clusters together represent the 2-hop cover.

Initially, the directed graph is converted to DAG and the reverse DAG is constructed. Reverse DAG is constructed by reversing the direction of edges of the DAG. Let "I" be the set having multi-interval encoding information of the nodes of DAG, i.e., for each node, postid and interval list is stored in "I". Let "It" be the set having multi-interval encoding information for the nodes of reverse DAG. Let $p(v_i)$ and $pt(v_i)$ be the postids of the node v_i in I and It respectively & let $I(v_i)$ & $It(v_i)$ denote the interval list of node v_i in "I" & "It" respectively where v_i is one of the vertices in V of the DAG G(V, E). The following pseudocode implements the steps of geometry-based approach and computes the 2-hop cover efficiently.

Algorithm 2-HopCover (I, It, V)

```
{ //size(V) returns the number of vertices of DAG in set V.
    n:=size(V);
    for i:=1 to n
    {
        for j:=1 to n, $v_i \neq v_j$
        {
        for k:=1 to size(I($v_i$)) //[$x_k$, $y_k$] is in I($v_i$)
        if ($x_k$<=p($v_j$)<=$y_k$) f[p($v_j$)][pt($v_i$)]:=1;
        }
    }//virtual reachability in 2D array f
    do
    {
    for m:=1 to n
   {m2:=Rect($v_m$); if(max<m2) {max:=m2; w:=$v_m$; }}
//get w in V with maximum count from  Rect(w).Let rf[i][j]
//be the array where reachability  of node 'w' is stored.
for i:=1 to n{  for j:=1 to n{if(rf[i][j]=f[i][j] & f[i][j]=1){
F.add ($p^{-1}$(i)); T.add ($pt^{-1}$(j)); f[i][j]:=0;}}}
H.add(S (F, w, T)); F.empty (); T.empty ();
} while (atleast one value in 2D array f is 1);
return H;
}
Rect(w)
{ count:=0;
 //I(w)={[s1,e1],s2,e2]..[sn,en]} & similarly for It(w).
        for k:=1 to size(I(w)) for i:=$s_k$ to $e_k$
        for l:=1 to size(It(w)) for j:=$s_l$  to $e_l$
        {
                rf[i][j]:=1; rf[p(w)][pt(w)]:=0;
                if(rf[i][j]=f[i][j]=1) count++;
        }
return count;
}
```

Pseudocode to compute 2-hop cover

In the pseudocode, the multi-interval encoding information is taken as input and the output returned is the 2-hop cover H. The explanation of the pseudocode along with the steps of the geometry-based technique is given below.

A. Construction of Virtual Reachability Map

For every node v_i, its reachability information is stored in a 2D array 'f' in the pseudocode defined as follows:

f[i][j]=1 if postid 'i' of a node (p[v_j]) lies in one of the intervals of v_i, and 'j' is the postid of the current node v_i (pt[v_i]) in reverse DAG.

This 2D array is termed as the virtual reachability map. This virtual reachability map contains the complete reachability information of the DAG.

B. Computation of Rectangular Map

Then, a rectangular map is created for each node v_i and stored in a temporary 2D array "rf". This map is created from the interval lists I and It of the current node v_i. Let I= [[s1, e1], [s2, e2],.[sk, ek]..[sn, en]]. Let It = [[$s1^1$, $e1^1$], [$s2^2$, $e2^2$],. .[sl, el]... [sn^n, en^n]] for v_i. The rectangular map "rf" computed in Rect(w) in the pseudocode is defined as follows:

For each interval [sk, ek] in I,
For each interval [sl, el] in It,
rf[i][j]=1 for all integers i such that i is in [sk, ek] and for all integers j such that j is in [sl, el].
rf[i][j]=0 if p[v_i]=i and pt[v_i]=j.

C. Derivation of 2-hop Cover

The procedure Rect(w) returns the total number of matching 1s of the virtual reachability map with "rf" of the node 'w' which is stored in variable "count". The node which has maximum value of "count" is selected. From the matching 1's of "rf" of such node 'w', a 2-hop cluster S (F, w, T) is derived. For each matching value, i.e. for each f[i][j]=rf[i][j]=1, where 1<=i<=n and 1<=j<=n, add the node v_i to F which has p[v_i]=i and add the node v_j to T which has pt[v_j]=j. Each 2-hop cluster formed can be visualized as a bipartite graph with 'w' as the center node of the bipartite graph. Thus, a 2-hop cluster is created which is added to 2-hop cover H. Then, assign to all the matching 1s in the virtual reachability map the value 0 and remove 'w'. This process of matching the rectangular maps with the virtual reachability map is repeated until no value in the virtual reachability map is 1. Thus, the 2-hop cover of DAG is computed.

The 2-hop clusters computed for the directed graph in Fig. 1 implementing the above pseudo-code are {[n1, n2, n4], n6, [n10, n8, n9]}, {[n1, n3], n5, [n10, n7, n8]}, {[n7, n8, n9], n10, []}, [], n1, [n1, n2, n3, n4]} & {[], n2, [n7]}. From the 2-hop cover, 2-hop labels can be derived for the nodes of the DAG. For each cluster S(Fw, w, Tw) , for each node 'u' in Fw, add 'w' to Lout (u) and for each node 'v' in Tw, add w to Lin (v).For instance, 2-hop labels derived for the directed graph in Fig. 1 are shown in Table II. Thus, 2-hop labels are constructed for the nodes of DAG. 2-hop labels for the other nodes of the directed graph are same as 2-hop labels of the maximal strongly connected nodes of the directed graph that represent them in DAG.

V. EXPERIMENTAL RESULTS

The graph data can be a large real XML document like DBLP data (http://dblp.uni-trier.de/xml/), or synthetic XML data like XMark [11] which are parsed to derive the directed data graph. XMark is a synthetic XML benchmark known for its irregular schema. There are elements that internally refer to other elements in the document which can be used to encode the XMark XML document as a directed graph with elements as nodes and parent-child relationships and referencing relationships as edges. The xmlgen tool of XMark constitutes a scaling factor which can be adjusted for generating XML documents with varying sizes.

We conducted our experiments on Dell laptop with 2.10 GHz processor and 3.0GB RAM running on Windows 7 in Java. 2-hop clusters are computed for XMark benchmark XML files using the efficient geometry based approach with the results outlined in the Table III. The last column in the Table III shows 2-hop labels which are computed from the 2-hop clusters (shown in third column of Table III). Fig. 3

TABLE III. EXPERIMENTED DATA AND RESULTS

XMark scaling factor	|V| of Directed Graph	|E| Of Directed Graph	No. of 2-hop clusters |S|	2-hop label size |L|
0.0001	372	438	109	891
0.0005	757	882	182	2444
0.0007	1143	1355	304	3237
0.0008	1158	1358	292	3419
0.001	1677	1961	480	4663

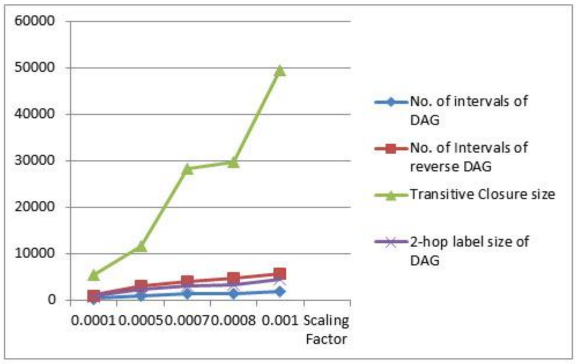

Figure 3. Comparison of Multi-interval Encoding size and 2-hop Labeling size to Transitive Closure size

shows significantly less 2-hop label size computed using our approach when compared to transitive closure size.

VI. CONCLUSION AND FUTURE WORK

The approach for computing 2-hop clusters is an efficient geometry-based approach which is tested on XMark XML files. From each 2-hop cluster $S(F_w, w, T_w)$, 2-hop reachability labels are constructed by adding center node 'w' to the label Lin(v) where 'v' is one of the nodes in T_w and adding center node 'w' to the label Lout(u) where 'u' is one of the nodes in F_w. The results shown in Fig. 3 of 2-hop labels computed show the significant amount of compression of transitive closure which indicates the efficiency of our approach. Using the 2-hop labels computed from 2-hop clusters, base relations and cluster-based index will be constructed that will be used in implementing the join-based algorithms [1] for efficiently solving the graph pattern matching problem.

REFERENCES

[1] J. Cheng, J. Xu Yu and P.S. Yu , "Graph Pattern Matching: A Join/Semijoin Approach", IEEE Transactions on Knowledge and Data Engineering, Vol. 23, No. 7, Jul. 2011, pp. 1006-1021, doi: 10.1109/TKDE.2010.169.

[2] J. Cheng, J. Xu Yu, X. Lin, H. Wang and P.S. Yu, "Fast Computation of Reachability Labeling for Large Graphs", Proc. International Conference of Extending Database Technology: Advances in Database Technology (EDBT '06), 2006.

[3] E. Cohen, E. Halperin, H. Kaplan and U. Zwick, "Reachability and Distance Queries via 2-Hop Labels", Proc. ACM-SIAM Symp. On Discrete Algorithms, (SODA 02), Jan. 2002, pp. 937-946.

[4] R. Agrawal, A. Borgida and H.V. Jagadish, "Efficient Management of Transitive Relationships in Large Data and Knowledge Bases", Proc. ACM Int'l Conf on Management Of Data, Jun. 1989, pp. 253-262.

[5] H. Wang, W. Wang, X. Lin and J. Li , "Coding-based Join Algorithms for Structural Queries on Graph-Structured XML Document", Vol. 11, No. 4, Springer 2008, pp. 485-510.

[6] R. Schenkel et al., "HOPI: An Efficient Connection Index for Complex XML Document Collections", Proc. Int'l Conf.(EDBT '04), 2004, pp. 237-255.

[7] R. Schenkel, A. Theobald and G. Weikum, "Efficient Creation and Incremental Maintenance of the HOPI Index for Complex XML Document Collections", Proc. 21st Int'l Conf. Data Eng. (ICDE '05), Apr. 2005, pp. 360-371.

[8] N. Bruno, N. Koudas and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching", Proc. ACM SIGMOD 2002, pp. 310-321.

[9] S. Chen et al., "Twig^{2}stack: Bottom-Up Processing of Generalized-Tree-Pattern Queries over XML Documents", Proc. Int'l conf. on Very Large DataBases,(VLDB'06), 2006, pp. 283-294.

[10] B. Bhargavi and K.P. Supreethi, "Graph pattern mining: A survey of issues and approaches", International Journal of IT and Knowledge Management, Vol.5, No. 2, pp. 401-407, Jul. 2012.

[11] A. Schmidt, F. Waas, M. Kersten, M.J. Carey, I. Manolescu and R. Busse, "XMark: A Benchmark for XML Data Management," Proc.28th Int'l Conf. Very Large Data Bases (VLDB '02), 2002.