Complex Query Processing on Web Graph: a Social Network Perspective

Susanta Mitra[1], Aditya Bagchi[2], A.K.Bandyopadhyay[3]
[1] International Institute of Information Technology
X – 1/8/3, Block EP
Salt Lake Electronics Complex, Sector-V
Kolkata 700091
India
susanta_mitra@yahoo.com

[2] Computer & Statistical Service Centre
Indian Statistical Institute
203, B.T. Road
Kolkata 700108
India
aditya@isical.ac.in

[3] Electronics & Telecommunication Engineering Department
Jadavpur University
Kolkata 700032
India
anupbandyopadhyay@hotmail.com

ABSTRACT: A social network represents a social community as a directed graph. Communication on the Web has given rise to social network formation like, Web Community, Referral System etc. An earlier effort has proposed a data model for such Web-based social network. Present paper discusses the relevant index structures for processing queries on a social network schema based upon the proposed data model. The paper has also provided evaluation of the structural operators proposed in the data model and discussed their efficacy with query examples.

Categories and Subject Descriptors
E.1 [Data Structures]; Graphs and networks **E.4 [Coding and Information Theory]** ; Data compaction and compression] **H3.4 [Systems and software]**; Information networks

General Terms
Web graphs, Social networking, Data compression

Keywords: Web Graph, Social Network, Query Processing, XB-tree, Operator Evaluation

Received 10 Aug. 2006; Revised and accepted 27 July 2007

1. Introduction

A social network is a social structure between actors (individuals, organization or other social entities) and indicates the ways in which they are connected through various social relationships like friendships, kinships, professional, academic etc. Some of the notable structural properties of a social network are connectedness between actors, reachability between a source and a target actor, reciprocity or pair-wise connection between actors with bi-directional links, centrality of actors or the important actors having high degree or more connections and finally the division of actors into sub-structures, like cliques or stronglyconnected components. The division of actors into strongly-connected components can be a very important factor for understanding a social structure, particularly the degree of cohesiveness in a community.

The formal representation of this pattern of relationships is a directed graph or digraph. In this graph, each member of a social community (people or other entities embedded in a social context) is considered as a node and communication (collaboration, interaction or influence) from one member of the community to another member is represented by a directed edge. In seventies Leinhardt first proposed the idea of representing a social community by a digraph [8]. A graph representing a social network has certain basic structural properties, which distinguishes it from other type of networks or graphs. The number of nodes in a social network can be very small representing a circle of friends or very large representing a Web community.

The diverse and distributed nature of *World-Wide-Web* has given rise to variety of research into the Web's link structure ranging from graph-theoretic studies (connectivity, reachability etc.) to community mining (like, discovering strongly connected structural components etc.). Recently, Web has played a major role in the formation of communities (*Cyber-communities or Web communities*) where members or people from different parts of the globe can join the community for common interest. Thus Web has become a good source of social networks. Structural similarities of Web with a social network help in studying different sociological behaviors of a Web community through applications of graph theory and social network analysis. These similarities lead towards a progress in knowledge representation and management on the Web [6]. Out of the many social network models on the Web, most commonly used one is called a *referral network*. In such a network, each node in the social network provides a set of links to its acquaintances that in turn become member nodes of the network. In the same way, these new nodes bring their acquaintances to the network again. This way the social network keeps on growing. So, the social network on the Web gives rise to an evolutionary graph. At any instant of time, when a query is raised on such an evolutionary graph, a snapshot of the concerned network, i.e. the node-edge structure at the instant of query, is considered for the purpose of query processing. The referral system has been both commercially (www.LinkedIn.com, www.Ryze.com, www.Tribe.com etc.) and academically [5] exploited.

1.1 Motivation

Lot of work has already been done for statistical analysis of a social network. The purpose is to study the properties of a social community. On the other hand, researchers in web-mining, web-based learning etc. have studied on the representation and navigation of Web graph structures. However, hardly any effort has been made to develop a data model for a Web graph so that queries can be made both on its structure as well as the node contents. Visualizing this need, authors of the present paper has proposed a data model representing a social network [10].

Since a Web graph representing a social network usually contains thousands of nodes and edges, it is necessary to adopt some methods of compression to efficiently handle such a graph for storage and querying. Authors of the present paper have already proposed the relevant methods of compression [2] and a data model, named SONSYS for such a compressed, Web-based social network [10]. SONSYS offers an Object Relational model supporting a Structural and a Node-based subsystem. It is evident that queries on a compressed graph proposed in SONSYS, need to access the underlying original graph as well. In addition, a query may include predicates related to node-based information as well. So, an elaborated index structure is necessary for efficient query processing on such a data model. An earlier effort has discussed the relevant index structures for both structural and node-based subsystems in a nutshell [9]. Primary motivation behind the present paper is to discuss the index structures in detail with the evaluation of the structural operators proposed in SONSYS. Considering complex queries, execution processes are then explained. Following the introduction, Section 2 discusses related work, Section 3 the data model, Section 4 the node-based indexing techniques and Section 5 the structural indexing techniques. Ultimately, Section 6 draws the conclusion.

2. Related Work

The graph data models proposed so far have mainly considered path-based queries or direct search of nodes and edges. A Web-based social network, on the other hand, considers many complex structures like cycles, nested cycles, strongly connected components etc. A few seminal works on Web graph models have now been discussed so that the relevance and efficacy of the proposed data model with its query facilities can be appreciated later.

The Web as a graph: measurement, models and methods [6] is one of the few initial works for studying the Web graph with a several hundred millions of nodes (html pages) and billions of directed edges (hyperlinks). The hyperlinks represent a source of a few sociological information. However, this application deals with one time data mining based analysis and does not provide a database platform for serving queries. Representing Web Graphs [11] has proposed a S-node representation scheme that combines graph compression with support for complex queries and local graph navigation for massive web graphs. However, only a few specific observed properties of the web graph could be exploited by this representation scheme. So, the system is designed to serve a set of dedicated queries. It is not a generic graph-based data model. Complex Queries over Web Repositories [12] has dealt with the complex web queries over massive Web graph. It has been shown that the S-node representation and the cluster-based optimization significantly reduce the query execution time. The query processing and the optimization techniques discussed here are based on the few inherent properties of the Web graph. However, it is not suitable for Web graphs used as a social network. Thus, it is apparent from the above discussion that none of the earlier research and development efforts have proposed any generic data model for Web graph represented as a social network.

3. Data Model

Web-based social network applications require execution of complex queries that involve combination of structural as well as node-based operations. The queries may involve searching of component structures and/or path-based search on the graph representing a social network. It may also need predicate based search on the properties of the nodes. The proposed generic Object-Relational (OR) data model can make structural, node-based, as well as composite queries on a Web-based social network application. This data model consists of two component subsystems; *Structural* subsystem and *Node-based* subsystem. Depending on application, these two subsystems give rise to two different sub-schemas under the objectrelational framework considered for modeling a Web-based social network. A 4-tier architecture proposed in [10] offers the mechanism to maneuver between the two subsystems and to answer the composite queries.

The subsequent sections describe the two subsystems in detail.

3.1 Structural Subsystem

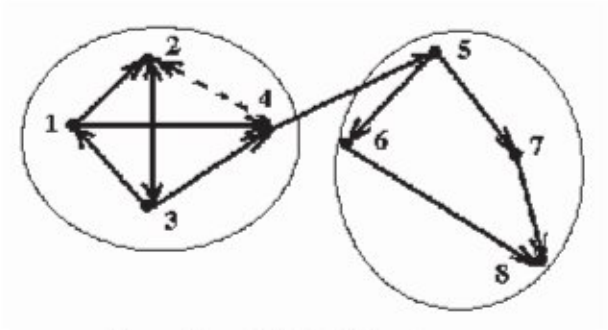


Figure 1 Sample Social Network

To understand the model, a sample social network, as shown in Fig.1, has been chosen. Although a few nodes and edges have been considered here, the explanation will soon show that even this small graph covers all the structural peculiarities of a social network on the Web.

As shown in Fig.1, the network initially had 4 nodes (1,2,3,4). Node 5 is the acquaintance of node 4. So, node 5 joined the net. In turn node 5 brought nodes 6 and 7 and they again brought node 8 in the network. This way the network keeps on growing. It is assumed that at the time of query, Fig.1 shows the current status of the network. Important structural components present in the network are described below. Formal definitions of the components are provided in [10].

- *Strongly-Connected-Component (SCC) :* The sample social network in Fig. 1 has two edge-types shown by farm and chain lines. Node sequence (1-2-3) represents a strongly connected component when same edge-types are considered, whereas (1-2-3-4) is a SCC considering both the edge-types.
- *Cycle :* Fig. 1 shows three cycles; (1-2-3-1), (2-3-4-2) and (2-3-2). Here cycles have been considered irrespective of the variation in edge-types. The cycles may even be nested. Cycle (2-3-2) is nested within the other two cycles, (1-2-3-1) and (2-3-4-2).
- *Reciprocal Edge:* In Fig. 1, (2-3-2) is a reciprocal edge.
- *Homogeneous Hyper-node:* In Fig. 1, (1-2-3) is a homogeneous hyper-node. It is a strongly connected component where all edges are of same type.

• *Heterogeneous Hyper-node:* In Fig.1, (1-2-3-4) is a heterogeneous hyper-node. It is a strongly connected component where all edges are not of same type. In Fig. 1, homogeneous hyper-node (1-2-3) is nested within heterogeneous hyper-node (1-2-3-4).

• *Hyper-edge:* If any node p outside a hyper-node H is connected to more than one node belonging to H with same edge-type and in the same direction, all such edges will be fused to only one edge as a hyper-edge. A hyper-edge may connect a hyper-node with a node or another hyper-node. After edge set {1,2,3} in Fig.1 fuses to the homogeneous hyper-node H-1 in Fig.2, edges (1,4) and (3,4) of Fig.1 fuses to the hyper-edge He-1 in Fig.2.

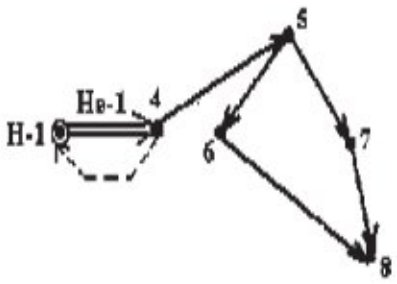


Figure 2. Augmented Social Network

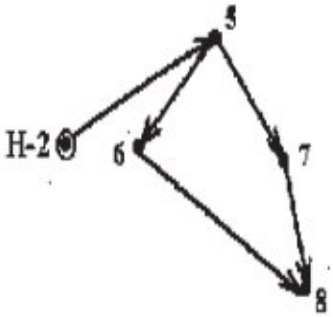


Figure 3. Final Augmentation

During pre-processing and compression phase as described in [2], each of these homogeneous and heterogeneous hyper-nodes are fused to a single node. In case of nesting of hyper-nodes [for example, homogeneous hyper-node (1-2-3) nested within heterogeneous hyper-node (1-2-3-4)] the outer-most one would ultimately be fused. Fig. 2 shows the compression when only homogeneous hyper-nodes are considered, whereas Fig. 3 shows the final augmentation after considering the heterogeneous hyper-nodes as well. At the end of compression process, as shown in Fig.3, the original digraph reduces to a directed acyclic graph (DAG) [2].

Definitions of the structural components offer an inherent hierarchy among them. A homogeneous hypernode may be nested within a heterogeneous hyper-node(1-2-3-4). Acycle may be nested within a hypernode. There can also be nested cycles. A cycle can again be decomposed to constituent nodes and edges. As discussed in [2], this nested structure again gives rise to a *Structural DAG* as shown in Fig.4.

This DAG essentially offers the complex object type hierarchy defined in the object-relational structural schema for representing a social network as explained in [10]. This DAG can be also be utilized to develop an index structure for structural query processing as discussed in Section 5.

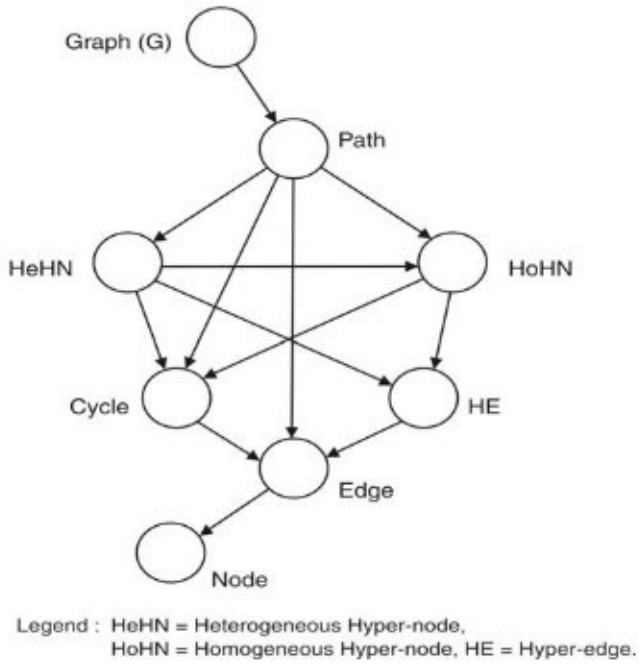


Figure 4. Structural DAG

3.2 Node-based Subsystem

Apart from the structural analysis, Web graph needs to be studied on the node-based information as well. For this purpose, the attributes or contents of a node or Web page are required to be properly represented, stored and indexed. A node-based schema can provide a convenient mechanism for representing Web page attributes. The information or contents of a page and a link are stored as attributes in two object types, *Page* and *Link* respectively, somewhat similar to Web relations in [13].

Each *Page* object is associated with a *p_ID* that represents the object-id as provided by a object-relationalsystem. It is synonymous to the corresponding node_id representing a Node object for that *Page* in the structural schema. Some of the other important attributes of a page are included in the object type with their meaning and the corresponding data types.

Each *Link* object is also associated with a *l_ID* that represents its object-id. Attribute *l_SrcID* is the identifier of the page in which a hyperlink originates and attribute *l_DestID* is the identifier of its target page. In otherwords, a *Link* object instance represents an edge of a Web graph where *l_SrcID* and *l_DestID* of the object refers to corresponding *p_ID*s of the pages it is linking. As a matter of fact, *l_SrcID* and *l_DestID* are basically structural information and would be available in structure-based sub-system as well. Here it has been provided for drawing similarity with the representation used in [13]. Some of the other important attributes of a page are included in the object type with their meaning and the corresponding data types.

The detail representation of structural and node-based schema for a social network has been provided in [10]. Since an object-relational framework has been used, there would be underlying relations representing the *Page* and *Link* object types. Such underlying relations are shown below :

Page (p_ID, p_Text, p_URL, p_Title, p_Topic, p_Language, p_Type, p_Length, p_Indegree, p_NumLinks, p_Domain, p_Host, p_PageRank)

Link (l_ID, l_SrcID, l_DestID)

The p_ID and l_ID shown in the relations are actually system supplied object-ids and not user specified. They have been shown here for the convenience of understanding the scenario.

These attributes have been borrowed from [13], which deals with different attributes of a Web page and the associated hyperlinks.

4. Node-based Indexing Techniques

SONSYS model employs separate indexing strategies for node-based and structural subsystems. Since structural subsystem covers the peculiarities of a graph structure that represents a social network, it asks for special operators to answer structural queries.

Operators for node-based queries, on the other hand, follow the standard ODMG specification. These operators employ standard indexing techniques used in an object relational system. In a social network, nodes can be categorized into different groups depending on certain node properties. This node categorization can then be exploited as a convenient index structure for efficient node-based query processing.

4.1 Node Categorization

It has been observed that there are certain node attributes on which the queries are made frequently. Domain of each of these important attributes can be partitioned into categories based on the domain values. For example, the attribute, *p_Domain* in *Page* can be categorized by the name of different domains like, *.edu*, *.org* etc. Categorization can also be made on the pages based on their language types. For example, attribute *p_Language* in *Page* can be used for categorization of page instances into different language types; *Engish, French or German*. This process of categorization leads to a semantic hierarchy of objects. In the parlance of object-orientation, it offers a class hierarchy. A sample node categorization hierarchy for the Web pages that related to research groups in universities of U.S. working on a few specialized areas is shown Fig.5.

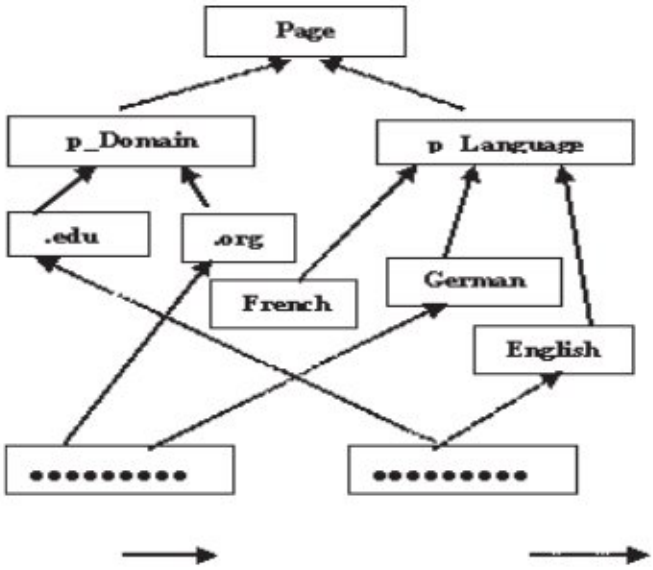


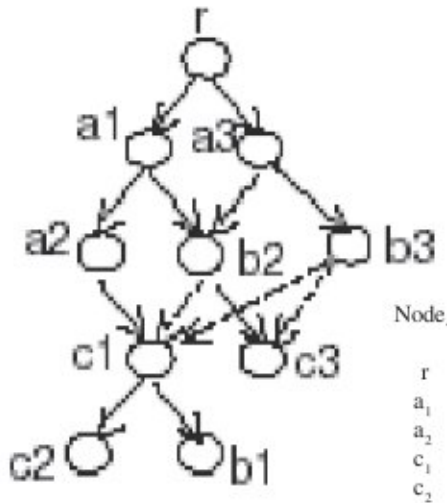
Figure . 5 Sample Node Categorization Hierarchy

This hierarchy can be used as an index structure for query processing. Value based indexing is quite common in query processing. Since the node-based sub-system is implemented as a standard objectrelational system, indexing facilities usually available in such systems have also been extended here. Usual B+ tree based indexes and hash based indexes have been provided as and when required.

5. Structural Indexing Techniques

This section describes a few novel indexing techniques for structural subsystem to accelerate search duringquery processing. The structural queries can be divided into two groups. One that involves queries to analyze the different structural relationships between the graph objects and hence requires traversal of the structural DAG. Other one employs path-based structural queries. So, specific operators are required for executing these two types of queries. Minimal set of operators required for these two types of queries have already been discussed in [10]. The present paper deals with their implementation with corresponding indexing strategies.

First type of query involves finding required query objects based upon the structural relationships like ancestors, descendants etc. between objects. For example, a cycle is an ancestor of nodes and edges, a hyper-node is an ancestor of a cycle etc. An index-based solution has been provided to efficiently search the structural DAG during query processing. Proposed indexing scheme covers two types of indexes; Spanning Tree index (ST-index) and Non-Spanning Tree index (NST-index). A sample structural DAG from [4] has been considered to explain the index structures. For tree and the non-tree cover of a DAG, technique described in [1] has been adopted. On the other hand, for node numbering, the interval encoding scheme of [4] has been applied. For this purpose, the sample DAG is traversed and each node is assigned an interval or an integer pair [start, end] according to the depth-first traversal. This DAG and the nodes with their corresponding intervals are shown in Fig.6. In this figure, the solid edges are the edges of the tree cover while dashed edges denote the non-tree edges or the remaining edges of the graph.



Node_id	Interval
r	(1,20)
a_1	(2,15)
a_2	(3,10)
c_1	(4,9)
c_2	(5,6)
b_1	(7,8)
b_2	(11,14)
c_3	(12,13)
a_3	(16,19)
b_3	(17,18)

Figure 6. Interval encoding representation of sample DAG.

The index structure similar to XB-tree [4] is meant for accessing nodes belonging to the spanning tree or the tree-cover and is named as *ST-tree*. A ST-tree corresponding to the sample DAG (Fig.6) is illustrated in Fig.7. In the ST-tree for the sample Structural DAG, each internal node contains interval(s) where each interval covers all the intervals of their respective descendant nodes and maintains pointer(s) to child node(s). As mentioned earlier, each interval has a start and an end

forming a [start, end] pair. Here a pair of ancestor and descendent nodes is related as,

ancestor.start < descendent.start and ancestor.end > descendent.end

For example, the intervals [2,15] and [16,19] are covered by the interval [1,20] of their common ancestor. Here the intervals are the search keys. Leaf nodes of ST-tree follow the structure of a B^+ tree.

Figure 7. ST-tree for sample DAG

NST-index

This new index structure, proposed in this paper, is for accessing anchor nodes. By definition [2], an anchor node may have more than one ancestor or more than one descendant or both. Also an anchor node is neither a source nor a sink node. Hence these nodes would be connected by non-tree edges and would be present in the *NST-tree* index structure. This structure has similarities with ST-index and is named as *NST-tree*. NST-tree corresponding to the sample DAG (Fig.6) is illustrated in Fig.8. In the NST-tree for the sample Structural DAG, each internal node represents an anchor node. Here also, each node maintains pointer(s) to its child node(s). However in NST-index, a pair of ancestor and descendent nodes is related as,

ancestor.start > descendent.start and ancestor.end > descendent.end

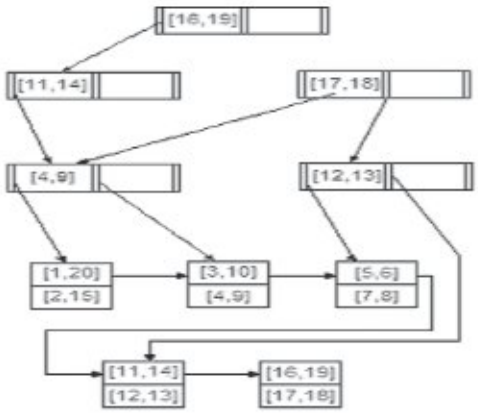


Figure 8. NST-tree for sample DAG

For example, the intervals [2,15] and [16,19] are covered by the interval [1,20] of their common ancestor. Here the intervals are the search keys. Leaf nodes of ST-tree follow the structure of a B+ tree.

NST-index

This new index structure, proposed in this paper, is for accessing anchor nodes. By definition [2], an anchor node may have more than one ancestor and hence a non-tree node. This structure has similarities with STindex and is named as *NST-tree*. NST-tree corresponding to the sample DAG (Fig.6) is illustrated in Fig.8. In the NST-tree for the sample Structural DAG, each internal node represents an anchor node. Here also, each node maintains pointer(s) to its child node(s). However in NST-index, a pair of ancestor and descendent nodes is related as,

ancestor.start > descendent.start and ancestor.end > descendent.end

For example, both [11,14] and [17,18] are the ancestors of [4,9]. So, [4,9] represents an anchor node and maintains the above ancestor-descendent relationship. Here again, the intervals are the search keys. Leaf nodes of NST-tree also follow the structure of a B_+ tree. While all nodes of the DAG are present in the ST-tree, only those nodes that are connected by non-tree edges as well, will appear in the NST-tree. So, with each node a flag is maintained to indicate whether it is present in the NST-tree.

5.1 Structural Query Operators

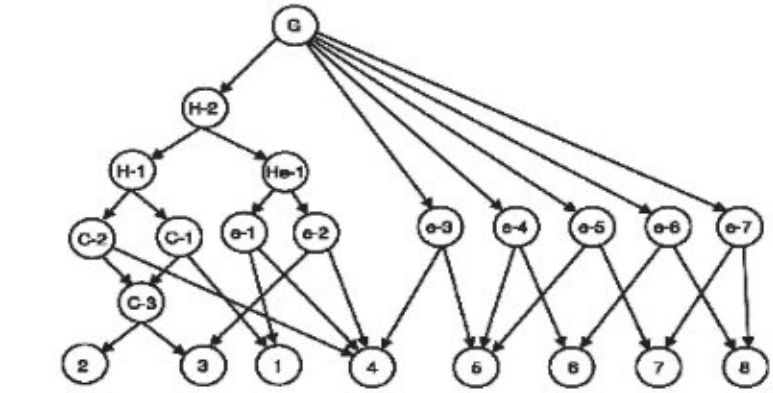
This section considers a set of new structural operators required for a social network. These operators are classified as *Fundamental* and *Derived*. They are shown in Table 1and Table 2 respectively. The relevance of these operators, their definitions, completeness and minimality have already been established in [10].

Operators	Purpose
Ancestor ($\subset$)	To find all ancestor structures of a referenced structure
Descendant ($\supset$)	To find all descendant structures of a referenced structure
Path (ρ)	To find all possible paths from source node

Table 1. List of fundamental operators

Operators	Purpose
Common_ Ancestor ($\wp$)	To find all ancestors common between two structures
Common_ Descendant(Υ)	To find all descendants common between two structures
Belongs_To (β)	To find all structures of a particular type that are ancestors to a specific type
Membership (μ)	To find all structures of a particular type that are descendants to a specific type
Enumeration (E)	To find all possible sub-paths between any two nodes
Reachability (R)	To return *true* if a node is reachable from a specified node

Table 2. List of derived operators



Legend : G = Graph, H-2 = Heterogeneous Hyper-node, H-1 = Homogeneous Hyper-node, He-1 = Hyper-edge, C-1 = Cycle (1-2-3-1), C-2 = Cycle (2-3-4-2), C-3 = Cycle (2-3-2)

Figure 9. Structural DAG for the sample Social Network

5.1.1 Evaluation of Operators

Considering the index structures defined above it is necessary to explain how the proposed set of structural operators exploit them during query processing. First, the operators that are used to form the queries to analyze the different structural relationships between the graph objects are considered. In order to explain the operators, structural DAG for the sample social network shown in Fig.1 and subsequently compressed in Fig.2 and Fig.3 is considered. This structural DAG is shown in Fig.9. A query is always is placed on the DAG generated after the compression process as explained in Fig.1 through Fig.3. However, the query may need to percolate down to the original graph to satisfy any node-based predicate. This percolation process needs to access the structural DAG as shown in Fig.9 for the sample social network.

Ancestor: Ancestor operator finds all the structures that are ancestor to a referenced structure as defined in complex structural object hierarchy of Fig.4. For example, the ancestor operation on cycle(2-3-2) (in Fig.1) represented by C-3 in Fig.9 returns cycles C-1 and C-2, hyper-nodes H-1 and H-2 and the original digraph (G) as the result set as shown in Fig.9.

A query may ask for all nodes that satisfy the predicate p and communicate with node x. So the query processor needs to access all the structural objects that are ancestors to node x. Breaking the encapsulation of those objects, the original nodes are then retrieved. Out of these nodes, the result set retains only those that satisfy the predicate p. Ancestor retrieval algorithm that exploits the ST and NST tree is detailed below:

Ancestor retrieval algorithm

```
procedure ANCESTOR(n, v)
// finds all ancestor structures n of the given structure v as obtained
from the structural DAG
    a_list = empty      // list of the ancestors of structure v,
                           initially empty
    int start(node);    // start number of a node in structural
                           DAG following the interval encoding
                           scheme explained in Fig.6
    int end(node);      // end number of a node in structural DAG
                           following the interval encoding
                           scheme explained in Fig.6
int out-count(node); // stores the current value of out-
                           degree  for each node
    int ST-outd(node);  // out-degree of a node in the ST-tree
nst(node) = true or false; // a boolean flag associated with each
                           node. It is true if the node has an  entry
                           in the NST-tree, otherwise false
visit(node) = false;    // a boolean flag for each node n in the
                           DAG. Initially false, turned true when
                                                       visited
// algorithm starts from the root of the structural DAG representing
the entire graph for the social network  and the default ancestor
for all nodes in the DAG
for an ancestor n of  v do // starting from root in order traversal
visits the nodes in ST-tree
begin
    out-count(n) = ST-outd(n);
    while out-count(n) <> 0 do
    begin
      out-count(n) = out-count(n)-1;
      visit descendent d of node n;
      if n <> d and visit(d) = false then
      begin
          if start(d) < start(v) and end(d) > end(v) then
          add d to a_list;
        if  nst(d) = true then
        begin
            if start(d) > start(v) and end(d) > end(v) then
            add d to a_list;
        end;
        ANCESTOR(d, v);
    end;
Returns a_list as the list of ancestors for structure v;
end;
```

Descendant: Descendant operator finds all the structures that are descendant to a referenced structure as defined in complex structural object hierarchy of Fig.4.

For example, the descendant operation on cycle(2-3-2) (in Fig.1) represented by C-3 in Fig.9 returns nodes 2 and 3 and other associated edges as its descendants.

Descendant retrieval algorithm

```
procedure DESCENDANT(n, v)
// finds all descendant structures n of the given structure v as obtained from the structural DAG
        d_list = empty        // list of the descendants of structure v, initially empty
        int start(node);      // start number of a node in structural DAG following the interval encoding scheme explained in Fig.6
int end(node);                // end number of a node in structural DAG following the interval encoding scheme explained in Fig.6
        int out-count(node);      // stores the current value of out-degree for each node
        int ST-outd(node);        // out-degree of a node in the ST-tree
        int NST-outd(node);       // out-degree of a node in the NST-tree
        nst(node) = true or false;  // a boolean flag associated with each node. It is true if the node has an entry in the NST-tree, otherwise false
        visit(node) = false;      // a boolean flag for each node n in the DAG. Initially false, turned true when visited
// starting from the given structure v, through in order traversal, algorithm visits all the descendant nodes in
   the ST-tree and augments the d_list
    begin
        if ST-outd(v) <> 0 then
        begin
          starting from v, in order traversal through ST-tree returns all the descendants of v in ST-tree;
          descendants obtained by the tree traversal are added to d_list;
       end;
       if nst(v) = true and NST-outd(v) <> 0 then
       begin
          starting from v, in order traversal through NST-tree returns all the descendants of v in ST-tree;
          descendants obtained by the tree traversal are added to d_list;
      end;
Returns d_list as the list of descendants for structure v;
end;
```

Since both ANCESTOR and DESCENDANT algorithms are tree search algorithms, considering minimum degree of the tree being t, complexity of the algorithms would be $O(\log_t n)$.

Common Ancestor Algorithm

```
procedure COMMON_ANCESTOR (u,v)        // finds the common set of ancestors between the structures u and v
begin
       ANCESTOR (n1, u);
       ANCESTOR (n2, v);
       CommonAncestors_list = { n1} Ç { n2}
end;
```

As explained earlier, ANCESTOR algorithm would have a complexity of $O(\log_t n)$. After getting the a_list for each structure u and v, a set intersection would provide the required of list of common ancestors. However, an intersection usually has a complexity $O(n^2)$. To reduce this complexity, the second ancestor algorithm ANCESTOR (n_2, v) has been modified. For each new ancestor retrieved for structure v, it is checked against the ancestor list already obtained for the first structure u. If a common member is found, it is pushed into the common_ancestor list. Since the members to a_list are added by tree traversal, they are ordered and so the complexity reduces to linear order, i.e. $n_1 + n_2$.

Common Descendant Algorithm

```
procedure COMMON_DESCENDANT (u,v)    // finds the common set of descendants between the structures u and v
begin
     DESCENDANT (n1, u);
     DESCENDANT (n2, v);
     CommonDescendants_list = { n1} Ç { n2}
end;
```

Implementation strategy for intersection operation is similar to the implementation done for Common Ancestor algorithm.

Belongs_To Algorithm

It finds all structures of a particular type that are ancestors to a specific type. It runs the ANCESTOR algorithm and then selects a particular type out of them. For example, the ancestor operation on cycle(2-3-2) (in Fig.1) represented by C-3 in Fig.9 returns cycles C-1 and C-2, hyper-nodes H-1 and H-2 and the original digraph (G) as the result set as shown in Fig.9. Now, if Belongs_To operation searches for hyper-nodes only, the result set will include only H-1 and H-2 discarding the others.

```
procedure BELONGS_TO (u, v)                    // searches for the
ancestors of u that are of type v
begin
        ANCESTOR (n, u);
        // returns all the ancestors n of u
        Select from n, where n-type = v-type;
end
```

Complexity of ANCESTOR algorithm has been discussed earlier. Selecting a specific structure type can be done in linear time.

Membership Algorithm

It find all structures of a particular type that are descendants to a specific type. It runs the DESCENDANT algorithm and then selects a particular type out of them.

For example, Membership operation on hyper-node H-1 will first find all the descendant structures under it as shown in Fig.9. However, if Membership operation searches for cycles, the resultant set will include only C-1, C-2 and C-3.

```
procedure MEMBERSHIP (u, v)                    /       /
searches for the descendants of u that are of type v
begin
        DESCENDANTS (n, v);
        // returns all the descendants n of u
        Select from n, where n-type = v-type;
end;
```

Complexity of DESCENDANT algorithm has been discussed earlier. Selecting a specific structure type can be done in linear time.

Reachability Algorithm

It finds whether a particular node in the DAG is reachable from another node. Reachability information is obtained from the node numbers. The encoding scheme is such that from the [start, end] values of two nodes the reachability information can be derived.

```
procedure REACHABLE (u,v)
Reachable = false;  // boolean flag that is set to 'true' if  node v is
                        reachable from node u;
                    // Initially it is set to 'false'.
begin
        if v is an ordinary node then
        begin
      if the interval, [start, end] associated with node v lies entirely
      within the interval, [start, end]  associated with node u i.e.
      u.start < v.start and u.end  > v.end  then
                Reachable = true;
        end
        else
        if v is an anchor node then     // here an anchor node has
in-degree > 1
        begin
        if v.start < u.start and v.end < u.end then
                Reachable = true;
        end;
end;
```

Path Search

Graph compression process for a social network converts the original graph to a DAG as explained from Fig.1 through Fig.3. As a result, the paths present in the original graph are altered. So, the path search process should be able to generate the original paths from the compressed DAG. This process of path storage and retrieval has been discussed in detail in [2]. Path search process returns all the simple paths in the graph.

Enumeration

While path search operation returns all the simple paths, Enumeration operator returns all paths between any two nodes u and v, where u and v are not necessarily a source and a sink respectively. It follows the same process as path search and the detail is given in [10].

6. Conclusion

This paper explains in detail the index structures for a object-relational data model designed for a Web graph used as a social network. This paper primarily addresses the different structural elements present in a Web graph. For ease of storage and retrieval, the graph is compressed to a DAG generating different structural elements like hyper-nodes, hyper-edges, cycles etc. These structural elements can then be arranged in a hierarchy, which provides an built-in index structure. Extending XB-tree, two types of tree index have been proposed. Following the method described in [1], ST-tree indexes the minimum tree cover for the structural DAG while NST-tree indexes the nodes through non tree edges. Different structural query operators are then studied to show how these operators are executed using the index structures. Complexity of the algorithms has also been studied.

However, this paper considers only a snapshot of a Web graph representing a social network. A social network is an evolutionary graph and its structure changes with time. Data modeling for such temporal changes will be considered later.

References

1. Agrawal, R., Borgida, A., Jagadish H.V(1989). Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. *In Proc. of ACM-SIGMOD*, pages 253-262.

2. Bhanu C.T., Mitra, S., Bagchi, A., Bandyopadhyay, A.K (2007). Pre-processing and Path Normalization of a Web Graph used as a Social Network. *Journal of Digital Information Management* 5 (5).

3. Bruno N., Koudas N., Srivastava D(2002). Holistic Twig Joins: Optimal XML Pattern Matching. *In Proc. of ACM-SIGMOD*.

4. Chen, L., Gupta, A., Kurul, E. M(2005). Efficient Algorithms for Pattern Matching on Directed Acyclic Graphs. *In: Proc. of IEEE International Conference on Data Engineering*.

5. Kautz. H., Selman., B, Shah. M(1997). The hidden Web. *AI Magazine*, 18 (2) 27–36.

6. Kleinberg, J. M., Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A (1999). Web as a graph: measurements, models and methods. *In: Proc. of Intl. Conf. on Combinatorics and Computing*, pages 1-18.

7. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A (2002). Web and Social Networks. *IEEE Computer*, 35 (11) 32-36.

8. Leinhardt, S.(1977). Social networks : a developing paradigm. *Academic Press*.

9. Mitra, S., Bagchi, A., Bandyopadhyay, A.K(2006). Complex Queries on Web Graph representing a Social Network. *In: Proc. of IEEE 1st International Conference on Digital Information Management (ICDIM)*, 424-430.

10. Mitra, S., Bagchi, A., Bandyopadhyay, A.K(2006). Design of a Data Model for Social Network Applications. *Journal of Database Management* (Accepted for publication).

11. Raghavan, S., Garcia-Molina, H(2003). Representing Web Graphs. *In: Proc. of IEEE Intl. Conf. on Data Engineering*.

12. Raghavan, S., Garcia-Molina, H(2003). Complex Queries over Web Repositories. *In Proc. of Intl. Conf. on Very Large Databases*.

13. Raghavan, S., Garcia-Molina, H(2003). Complex Queries over Web Repositories. *Technical report, Stanford University.* http://dbpubs.stanford.edu/pub/2003-11.

14. Rao, A.R.,Bandyopadhyay, S (1987). Measures of reciprocity in a social network. *Sankhya : The Indian Journal of Statistics,* Series A, 49, 141-188.

15. Rao, A.R., Bandyopadhyay, S., Sinha, B.K., Bagchi, A., Jana, R., Chaudhuri, A. and Sen, D (1998). Changing Social Relations – Social Network Approach, Technical Report. *Survey Research and Data Analysis Center, Indian Statistical Institute.*

Biographies of Authors

Susanta Mitra is a Professor and Head of Department of Computer Science & Engineering and IT at Meghnad Saha Institute of Technology, Kolkata, India. He has served in software industries for more than 17 years before joining IIIT. His areas of research for Ph.D. from Jadavpur University, India, included Web Graph analysis and data modeling, social networking and object-relational database technology. His current research interests are in Web data management, Web data mining, data structure and algorithms. He has published papers in international conferences that include publications from IEEE, Elsevier Science. His papers have also been selected for publication in renowned international journals.

Aditya Bagchi is the Chief, Consultancy and Development (a professor's post) at the Computer & Statistical Service Center, Indian Statistical Institute, Kolkata, India. Prof. Bagchi received his Ph.D (Engg.) from Jadavpur University in 1987. He has served in Tata Consultacy Services, Tata Burroughs Ltd, CMC Ltd and Regional Computer Center, Kolkata before joining the Indian Statistical Institute in 1988. Prof. Bagchi has also served as visiting faculty of Jadavpur University and BE Collage (Bengal Engineering and Science University), India. He has served as visiting scientist at the San Diego Super Computer Center, University of California, San Diego, USA and at the Center for Secure Information Systems, George Mason University, Virginia, USA. Prof. Bagchi is a Senior Member of Computer Society of India, member of ACM Sigmod and IEEE Computer Society. His research interests include Access Control and Trust Negotiation algorithms, developing new measures for Association Rule mining and application specific Data Modeling.

Arup Kumar Bandyopadhyay received the B.E. Tel. E., M. E. Tel. E and Ph.D. (Engg.) degrees from Jadavpur University, Kolkata, India in 1968, 1970 and 1983 respectively. From 1970 to 1972 he worked with the Microwave Antenna System Engineering Group of the Indian Space Research Organization. In 1972 he joined the Department of Electronics and Telecommunication Engineering, Jadavpur University, where he is currently a professor. His research interests include program verification and construction of distributed systems.