Graph Encoding and Transitive Closure Representation

Yangjun Chen
University of Winnipeg, Canada

INTRODUCTION

Composite objects represented as directed graphs are an important data structure that require efficient support Web and document databases (Abiteboul, Cluet, Christophides, Milo, Moerkotte, & Simon, 1997; Chen & Aberer, 1998, 1999; Mendelzon, Mihaila, & Milo, 1997; Zhang, Naughton, Dewitt, Luo, & Lohman, 2001), CAD/ CAM, CASE, office systems, and software management. It is cumbersome to handle such objects in relational database systems when they involve ancestor-descendant relations (or say, reachability relations). In this article, we present a new graph encoding based on a tree labeling method and the concept of branchings that are used in the graph theory for finding the shortest connection networks. A branching is a subgraph of a given digraph that is in fact a forest, but covers all the nodes of the graph. Concretely, for a DAG G (directed acyclic graph) of n nodes, the space needed for storing its transitive closure can be reduced to $O(b \cdot n)$, where b is the number of the leaf nodes of G's branching. Such a compression is, however, at the expense of querying time. Theoretically, it takes $O(\log b)$ time to check whether a node is reachable from another. The method can also be extended to digraphs containing cycles.

BACKGROUND

A composite object can be generally represented as a directed graph (digraph for short). For example, in a CAD database, a composite object corresponds to a complex design, which is composed of several subdesigns. Often, subdesigns are shared by more than one higher-level design, and a set of design hierarchies thus forms a directed acyclic graph (DAG). As another example, the citation index of scientific literature, recording reference relationships between authors, constructs a directed cyclic graph. As a third example, we consider the traditional organization of a company, with a variable number of manager-subordinate levels, which can be represented as a tree hierarchy.

In a relational system, composite objects must be fragmented across many relations, requiring joins to gather all the parts. A typical approach to improving join efficiency is to equip relations with hidden pointer fields for coupling the tuples to be joined. The so-called *join index* is another auxiliary access path to mitigate this difficulty. Also, several advanced join algorithms have been suggested, based on hashing and a large main memory. In addition, a different kind of attempts to attain a compromise solution is to extend relational databases with new features, such as *clustering* of composite objects, by which the concatenated foreign keys of ancestor paths are stored in a primary key. Another extension to relational system is *nested relations* (or NF^2 relations). Although it can be used to represent composite objects without sacrificing the relational theory, it suffers from the problem that subrelations cannot be shared. Moreover, recursive relationships cannot be represented by simple nesting because the depth is not fixed. Finally, *deductive database*s and *object-relational databases* can be considered as two quite different extensions to handle this problem (Chen, 2003; Ramakrishnan & Ullman, 1995).

In the past decade, another kind of research has been done to avoid *join* operation based on *graph encoding*. In this article, we provide an overview on most important techniques in this area and discuss a new encoding approach to pack "ancestor paths" in a relational environment (Chen, 2004; Chen & Cooke, 2006). It needs only $O(e \cdot b)$ time and $O(n \cdot b)$ space, where b is the number of the leaf nodes of the graph's branching. This computational complexity is better than any existing method for this problem, including the graph-based algorithms (Bender, Farach-Colton, Pemmasani, Skiena, & Sumazin, 2004; Schmitz, 1983), the graph encoding (Abdeddaim, 1997; Bommel & Beck, 2000; Zibin & Gil, 2001), and the matrix-based algorithms (La Poutre & Leeuwen, 1988).

TREE LABELING

In this section, we mainly discuss the concept of tree labeling, based on which our algorithm is designed. For any directed tree T, we can label it as follows. By traversing T in *preorder*, each node v will obtain a number $pre(v)$ to record the order in which the nodes of the tree are visited. In a similar way, by traversing T in *postorder*, each node v will get another number $post(v)$. These two numbers can be used to characterize the reachabilities of nodes as follows.

- **Proposition 1:** Let v and v' be two nodes of a tree T. Then, v' is a descendant of v iff $pre(v') > pre(v)$ and $post(v') < post(v)$.

- **Proof:** See Exercise 2.3.2-20 in Knuth (1969).

The following example helps for illustration.

- **Example 1:** See the pairs associated with the nodes of the directed tree shown in Figure 1. The first element of each pair is the preorder number of the corresponding node and the second is its postorder number. Using such labels, the reachabilities of nodes can be easily checked. For instance, by checking the label associated with b against the label for f, we know that b is an ancestor of f in terms of Proposition 1. We can also see that since the pairs associated with g and c do not satisfy the condition given in Proposition 1, g must not be an ancestor of c and *vice versa*.

Let (p, q) and (p', q') be two pairs associated with nodes u and v. We say that (p, q) is subsumed by (p', q'), denoted $(p, q) \prec (p', q')$, if $p > p'$ and $q < q'$. Then, u is a descendant of v if (p, q) is subsumed by (p', q').

GRAPH DECOMPOSITION AND COMPUTATION OF TRANSITIVE CLOSURES

Now we discuss how to recognize the ancestor-descendant relationships w.r.t. a general structure: a DAG or a graph containing no cycles. First, we discuss the concept of branching in 4.1. Then, we address the problem of DAGs in 4.2. Next, cyclic graphs are discussed in 4.3.

Branchings of DAGs

What we want is to apply the technique discussed above to a DAG. For this purpose, the concept of *branchings* needs to be first specified.

- **Definition 1:** (*branching* (Tarjan, 1977)) A subgraph $B = (V, E')$ of a digraph $G = (V, E)$ is called a branching if it is cycle-free and $d_{\text{indegree}}(v) \leq 1$ for every $v \in V$.

Clearly, if for only one node r, $d_{\text{indegree}}(r) = 0$, and for all the rest of the nodes, v, $d_{\text{indegree}}(v) = 1$, then the branching is a directed tree with root r. Normally, a branching is a set of directed trees. Now, we assign each edge e a same cost (e.g.,

Figure 1. Tree labeling

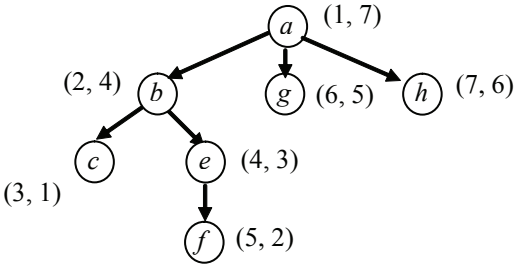

Figure 2. A DAG and its branching