Graph Encoding and Recursion Computation

Yangjun Chen
University of Winnipeg, Canada

INTRODUCTION

It is a general opinion that relational database systems are inadequate for manipulating composite objects that arise in novel applications such as 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. Especially, when recursive relationships are involved, it is cumbersome to handle them in relational databases, which sets current relational systems far behind the navigational ones (Kuno & Rundensteiner, 1998; Lee & Lee, 1998). To overcome this problem, a lot of interesting graph encoding methods have been developed to mitigate the difficulty to some extent. In this article, we give a brief description of some important methods, including analysis and comparison of their space and time complexities.

BACKGROUND

A composite object can be generally represented as a directed graph (digraph). 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 attempt 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).

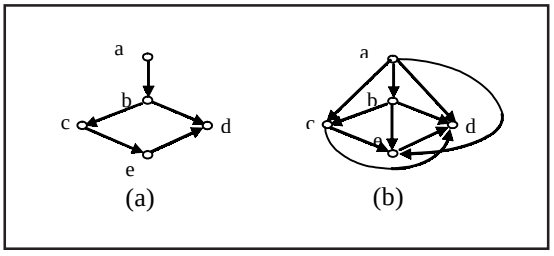
In the past decade, a quite different kind of research has also been done to avoid *join* operations based on *graph encoding*. In this article, we provide an overview on most important techniques in this area and discuss a new encoding approach to pack "ancestor paths" in a relational environment. It needs only $O(e \cdot b)$ time and $O(n \cdot b)$ space, where b is the breadth of the graph, defined to be the least number of disjoined paths that cover all the nodes of a graph. This computational complexity is better than any existing method for this problem, including the graph-based algorithms (Schmitz, 1983), the graph encoding (Abdeddaim, 1997; Bommel & Beck; Zibin & Gil, 2001) and the matrix-based algorithms (La Poutre & Leeuwen, 1988).

RECURSION COMPUTATION IN RELATIONAL DATABASES

We consider composite objects represented by a digraph, where nodes stand for objects and edges for parent-child relationships, stored in a binary relation. In many applications, the transitive closure of a digraph needs to be computed, which is defined to be all ancestor-descendant pairs. For instance, the transitive closure of the graph in Figure 1(a) is shown in Figure 1(b).

In this article, we mainly overview the graph encoding in a relational environment. The following is a typical structure to accommodate part-subpart relationships (Cattell & Skeen, 1992):

Figure 1. A graph and its transitive closure



- Part(Part-id, ...)
- Connection(Parent-id, Child-id, ...)

where Parent-id and Child-id are both foreign keys, referring to Part-id. In order to speed up the recursion evaluation, we will associate each node with a pair of integers, which helps to recognize ancestor-descendant relationships.

In the rest of the article, the following three types of digraphs will be discussed.

(i) Tree hierarchy, in which the parent-child relationship is of one-to-many type; that is, each node has at most one parent.
(ii) Directed acyclic graph (DAG), which occurs when the relationship is of many-to-many type, with the restriction that a part cannot be sub/superpart of itself (directly or indirectly).
(iii) Directed cyclic graph, which contains cycles.

Later we will use the term *graph* to refer to the *directed graph*, since we do not discuss non-directed ones at all.

RECURSION WITH RESPECT TO TREES

Perhaps the most elegant algorithm for encoding is *relative numbering* (Schmitz, 1983), which guarantees both optimal encoding length of log*n* bits and constant time recursion tests for trees. Consider a tree *T*. By traversing *T* in *postorder*, each node *v* will obtain a number (it can be integer or a real number) *post*(*v*) to record the order in which the nodes of the tree are visited. A basic property of postorder traversal is

$$post(v) = \max\{\, post(u) \mid u \in \text{descendant}(v)\}.$$

Let $l(v)$ be defined by

$$l(v) = \min\{\, post(u) \mid u \in \text{descendant}(v)\}.$$

Then, a node *u* is a descendant of *v* if $l(v) \le post(u) \le post(v)$.

In terms of this relationship, each node *v* can be encoded by an interval $[l(v), post(v)]$ as exemplified by Figure 2(a).

Another interesting graph encoding method is discussed in Knuth (2003), by which each node is associated with a pair $(pre(v), post(v))$, where $pre(v)$ represents the preorder number of *v* and can be obtained by traversing *T* in a preorder. The pair can be used to characterize the ancestor-descendant relationship as follows.

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

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

Figure 2. Labeling a tree