

Graph Representation Comparative Study

Nabil Arman
Assistant Professor
Palestine Polytechnic University
PO Box 198, Hebron, Palestine
Email: nabil_arman@yahoo.com
Tel. No.: 0097022228912

Abstract: *There are a number of graph representation schemes that have been used in graph algorithms. In this short article, a comparison is conducted among three major representations of directed graphs to illustrate the main advantages and disadvantages of each representation scheme. The reason for conducting this study is to show that one of the schemes is overlooked despite the fact that it has more information than the other schemes and this information is very useful in improving the performance of many graph algorithms.*

Keywords: *Graph Representation, Path Matrix Representation*

1. Introduction

Graph algorithms have attracted a large amount of research efforts due to the important role of these algorithms in many application domains. The efficiency of many graph algorithms is largely dependent on the underlying graph representation scheme. Many graph algorithms are well-documented in the literature, including specialized texts and research papers.

In this paper, three directed graph representation schemes are compared, namely Path Matrix, Adjacency Matrix, and Adjacency Lists. The remainder of the paper is organized as follows. Section 2 explains the three representation schemes. The comparison criteria are explained in section 3. The conclusion is presented in section 4.

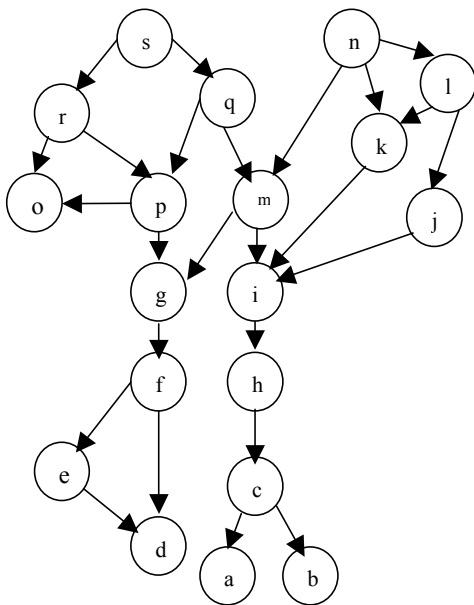
2. Graph Representation Schemes

In this section, the directed graph representation schemes are presented. The Path Matrix scheme is explained in detail, since it is not very popular compared to the other two schemes, which are well-known.

2.1 The Path Matrix Graph Representation

The path matrix is a special matrix/structure that has been used in answering the generalized forms of partially and fully instantiated same generation queries in deductive databases [1,2] and in computing the transitive closure of a database relation [4]. In this matrix, the rows represent some paths in the graph starting from the roots/source vertex to the leaves. Basically, depth-first search is used to create the paths of the graph. Instead of storing every vertex in all paths, the common parts of these paths can be stored only once to avoid duplications. If two paths $P_1 = \langle a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m \rangle$ and $P_2 = \langle a_1, a_2, \dots, a_n, c_1, c_2, \dots, c_l \rangle$ have the common parts $\langle a_1, a_2, \dots, a_n \rangle$, then P_1 and P_2 can be stored in the two consecutive

rows of the matrix as $\langle a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m \rangle$ and $\langle \dots, n \text{ empty entries} \dots, c_1, c_2, \dots, c_l \rangle$, where the first n entries of the second row are empty. To prevent the duplicate storage of the vertex in the matrix, a different technique is used; for the first visit to the vertex, it is entered into the matrix and the coordinates of its location is recorded. On subsequent visits, instead of entering the vertex itself, its coordinates are entered into the matrix (a pointer to the already stored vertex). In this way, only a single copy of each of the graph's vertex is guaranteed to be entered in the matrix. Moreover, there will be only one entry (either a vertex or a pointer) in the matrix for each edge in the graph. In Figure 1 (b), the matrix representation of the graph given in Figure 1 (a) is presented. In that graph, there are 25 edges, and in its matrix representation there are $25+2=27$ nonempty entries in the matrix (another two entries for the vertex s and n). An important advantage of this matrix structure is that it stores a path from each root to each vertex that is reachable from that root.



	0	1	2	3	4	5	6
0	s	r	o				
1			p	0,2			
2				g	f	e	d
3						2,6	
4		q	1,2				
5			m	2,3			
6				i	h	c	a
7							b
8	n	5,2					
9		k	6,3				
10		l	9,1				
11			j	6,3			

(a) Graph Form

(b) Matrix Representation

Figure 1. Directed Graph in (a) Graph Form (b) Matrix Representation

In the implementation of this sparse matrix, the empty entries are not stored explicitly. The matrix can be stored sequentially row by row as shown in Figure 2. For each row, storing the column number, as shown in Figure 3, of its first non-empty entry and the sequence of non-empty entries in the row is sufficient. Thus, the size of the stored matrix is much smaller than the original relation and matrix.

0,0	0,1	0,2	1,2	1,3	2,3	2,4	2,5	2,6	3,5	4,1	4,2	5,2	5,3	6,3
s	r	o	p	0,2	g	f	e	d	2,6	q	1,2	m	2,3	i

6,4	6,5	6,6	7,6	8,0	8,1	9,1	9,2	10,1	10,2	11,2	11,2
h	C	a	b	N	5,2	k	6,3	l	9,1	j	6,3

Figure 2. The Matrix as Linear Array

Row No.	Row Beginning	Row End	Row End Entry
0	0	2	o
1	2	3	<0,2>
2	3	6	d
3	5	5	<2,6>
4	1	2	<1,2>
5	2	3	<2,3>
6	3	6	a
7	6	6	b
8	0	1	<5,2>
9	1	2	<6,3>
10	1	2	<9,1>
11	2	3	<6,3>

Figure 3. Matrix Representation Row Beginnings and Endings

2.2 The Adjacency Matrix Graph Representation

The most straightforward scheme of directed graph representation is the so-called adjacency matrix representation. A V -by- V array of Boolean values is maintained, with $a[x,y]$ set to true if there is an edge from vertex x to vertex y and false otherwise. The adjacency matrix representation for the graph in Figure 1 (a) is shown in Figure 4. (1 means *true* and 0 means *false*). The first step in representing a graph is to map the vertex names to integers between 1 and V to make it possible to access information using array indexing [3].

$x \setminus y$	s	r	o	p	g	f	e	d	q	m	i	h	c	a	b	n	k	l	j
s	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
r	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
q	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
k	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
j	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 4. Adjacency Matrix Representation

2.3 The Adjacency List Graph Representation

In this representation, all the vertices connected to each vertex are listed on an *adjacency list* for that vertex. This can be easily done with linked lists. The linked lists are built as usual, with artificial nodes for the beginning of the lists kept in an array *adj* indexed by vertex. To add an a directed edge connecting x to y to this representation of the graph, we add y to x 's adjacency list [3].

3. Comparison Criteria

In this section, the three representation schemes are compared according to the following criteria:

a) Intelligence and Pruning

Algorithms that use the path matrix representation can benefit from the properties of this representation to prune or bound the traversing of the vertices in the matrix, since it terminates the search once a pointer goes beyond the vertex that the algorithm tries to reach and then backtracks from there. On the other hand, algorithms that use the adjacency matrix or adjacency list representations may explore irrelevant parts of the graph. Assume we are trying to determine whether there is a path from vertex v_1 to vertex v_2 . If DFS starts with v_1 , then it will consider all vertices reachable from that vertex until it reaches v_2 (if v_2 is reachable). However, if DFS starts with v_1 and uses the path matrix representation, then it will consider vertices reachable from that vertex as long as no vertex coordinates goes beyond the coordinates of vertex v_2 or until it reaches v_2 (if v_2 is reachable).

b) Database Usage (Vertex Clustering)

An important advantage of the path matrix representation is that it stores a path from the source vertices to all vertices reachable from that source vertex. Clustering vertices around source vertices reduces the number of page I/Os necessary to process data needed in many graph algorithms. On the other hand, the way adjacency matrix representation keeps its information increases dramatically the number of page I/Os necessary to process data in graph algorithms.

c) Path Ordering

In the path matrix representation, paths have implicit order according to the rows they appear in. On the other hand, neither adjacency matrices nor adjacency lists have such order. This ordering plays a crucial role in pruning the search of graphs once a certain path number has been exceeded.

d) Space Requirements

As explained before, in the implementation of the path matrix representation, the empty entries are not stored explicitly. The matrix can be stored sequentially row by row as shown in Figure 2. For each row, storing the column number, as shown in Figure 3, of its first non-empty entry and the sequence of non-empty entries in the row

is sufficient. Thus, the size of the stored matrix is much smaller than the original graph and matrix. The space requirement is actually proportional to $|E|+|s|$ where $|E|$ is the number of edges and $|s|$ is the number of source nodes ($|s|$ is generally a constant).

However, the adjacency matrix representation needs space proportional to $|V|^2$ and the adjacency list representation needs space proportional to $|V|+|E|$. Therefore, the space needed by path matrix representation is less than the space needed by the other two representation schemes.

e) Vertex Skipping in Searching

In certain path algorithms, such as finding the path length between two vertices, or finding the maximum/minimum path length between two vertices, or even checking the path existence between two vertices, path matrix representation enables these algorithms to skip many vertices on different paths, instead of considering all vertices on these paths, using the row beginnings and row ends to determine the paths or path lengths.

4. Conclusion

This paper presents a comparative study of graph representation schemes. Each representation scheme is explained briefly and its main advantages and disadvantages is presented. The criteria are useful in choosing the appropriate representation scheme for different internal or external algorithms. The paper also highlights the advantages of path matrix representation scheme since it is overlooked in graph theory related algorithms. As a next step in this research project, we intend to further study these schemes by conducting some simulations for different algorithms.

References

- [1] Arman, N., "An Efficient Algorithm for the Generalized Partially Instantiated Same Generation Query in Deductive Databases," *The International Arab Journal of Information Technology*, pp. 142-146, Vol. 1, No. 1, January 2004.
- [2] Arman, N., " An Intelligent Algorithm for the Generalized Fully Instantiated Same Generation Query in Deductive Databases," *Proceedings of the 4th International Arab Conference on Information Technology (ACIT'2003)*, pp. 224-228, December 20-23, 2003, Arab Academy for Science and Technology, Alexandria, Egypt.
- [3] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001), *Introduction to Algorithms*, McGraw-Hill Book Company.
- [4] Toroslu, I, Qadah, G., and Henschen, L., An Efficient Database Transitive Closure Algorithm, *Journal of Applied Intelligence* 4, pp. 205-218, 1994.