Computing the Transitive Closure of Symmetric Matrices
Anestis A. Toptsis, Clement T. Yu, and Peter C. Nelson
Dept. of Electrical Engineering
and Computer Science
University of Illinois
Chicago, IL 60680, USA

ABSTRACT

Certain real life binary relations are symmetric (map point connections, "if" mathematical statements, multiprocessor links). Such relations can be represented by symmetric 0-1 matrices. Algorithms which take advantage of the symmetry when acting on such matrices, are more efficient than algorithms that are "good for all cases" by assuming a generic (non-symmetric) matrix. No algorithm, to our knowledge, focusing on symmetric matrices has been designed up to date for the computation of the transitive closure. In this paper, four algorithms - G, Symmetric, 0-1-G, 1-Symmetric - are given for computing the transitive closure of a symmetric binary relation which is represented by a 0-1 matrix. Algorithms G and 0-1-G pose no restriction on the type of the input matrix, while algorithms Symmetric and 1-Symmetric require it to be symmetric. These four algorithms are compared to Warren's algorithm in terms of the number of page faults incurred. Experimental results indicate that the new algorithms (with the exception of algorithm G) are about 2 times faster than Warren's algorithm for sparse matrices, 10 to 100 times faster for dense matrices, and about 1.4 times faster for medium dense matrices.

Keywords: Transitive closure, symmetric matrix, binary relation.

1 Introduction

Let G=(V,E) be a directed graph, where V is the set of vertices (nodes) and E is the set of edges of the graph G. The transitive closure of G is a graph G'=(V,E'), where E'=E U \{e=(ij)\} - there is a directed path from i to j of the form i -> ... -> j in G). In terms of matrices, we can represent G by its incidence matrix M, i.e. M(ij)=1 if there is an edge from i to j, and M(ij)=0 otherwise. Then the transitive closure of M is a matrix M' such that M'(ij)=1 if there is a directed path from node i to node j in G, and M'(ij)=0 otherwise. A matrix M is symmetric if M(ij)=M(ji) for all i and j. A binary relation R can be represented by a 0-1 matrix M in which M(ij)=1 if iRj, and M(ij)=0 otherwise. Symmetric matrices can be considered to be adjacency matrices of undirected graphs (or directed graphs where the existence of the edge i -> j implies the existence of the edge j -> i and visa versa). Such graphs are used to represent symmetric binary relations.

Finding transitive closure has received considerable attention [1] [3] [2] [4] [6] [7] [8] [10] [11] [12] [13] [14] [15] [16] [17]. It appears so far in the literature, all algorithms that compute transitive closure assume the underlying binary relation to be a general (non-symmetric) directed graph. Although these algorithms also work for undirected graphs (since any undirected graph can be perceived as a special case of a directed graph), we maintain that algorithms especially designed for undirected graphs are better suited for applications that yield symmetric adjacency matrices, like the ones in the examples above. Algorithms that take advantage of the symmetry are presented in this paper.

It seems, so far, that two different kinds of input are the most popular, among all inputs. One is the 0-1 matrix representation described above, and the other is the successor list representation. For example, if the immediate successors of node 1 in the graph are the nodes 3, 4, 5, the successor list of 1 is the list [3,4,5]. The successor list representation is more efficient in space and time if the input graph has very few edges, while the 0-1 matrix representation is more efficient in space and time for graphs with more edges (see Appendix). So far, Warren's algorithm [16], is the best algorithm in terms of page faults, accepting as input a 0-1 matrix, and the algorithms in [8] are the best among those algorithms that accept as input successor lists. In this paper we choose to adopt the 0-1 matrix representation as the format of input to be accepted by our algorithms and compare the performance of 4 algorithms with Warren's algorithm. We assume as in [16] that only a few rows (or columns) of a matrix can be held in main memory. A major factor to be considered in the course of the computation is the number of disk access requests (page faults) incurred [11]. Experimental results indicate that the new algorithms (with the exception of algorithm G) are about 2 times faster than Warren's algorithm for sparse matrices, 10 to 100 times faster for dense matrices, and about 1.4 times faster for medium dense matrices.

In section 2, we give a short review of Warren's algorithm [16]. In section 3, algorithm G is presented. Algorithm G is applicable to any, not necessarily symmetric, matrix. However, it may not be efficient in the number of page faults. Therefore, in section 4, we introduce algorithm Symmetric which is applicable to symmetric matrices. Section 5 discusses algorithms 0-1-G and 1-Symmetric which take advantage of sparseness and denseness of matrices. Algorithm 0-1-G is applicable to any, not necessarily symmetric matrix, while algorithm 1-Symmetric is applicable to symmetric matrices. Experimental results comparing these four algorithms with Warren's algorithm, are given and analyzed in section 6. Finally, in section 7, we summarize our results, and indicate problems for further research.

1Current address: Dept. of Computer Science and Mathematics, York U., Atkinson College, Toronto, ONT, Canada, M3J-1P3
2 Warren's algorithm

Warren [16] gave an algorithm for computing the transitive closure of a binary relation R that is represented by a 0-1 $n \times n$ matrix $M$.

Warren's algorithm is as follows:

1. for $i = 2$ to $n$ /* 1-st pass */
   2. for $j = 1$ to $i-1$
   3. if $M(i,j) = 1$ then
   4. $row(i) = row(i) \lor row(j)$;
   5. for $i = 1$ to $n-1$ /* 2-nd pass */
   6. for $j = i+1$ to $n$
   7. if $M(i,j) = 1$ then
   8. $row(i) = row(i) \lor row(j)$;

Warren's algorithm makes two passes over the matrix, scans by rows and incurs fewer page faults than Warshall's algorithm [17]. In the first pass, it scans all entries below the main diagonal and updates each row using its preceding rows. In the second pass it scans all entries above the main diagonal and updates each row using its successor rows.

3 Algorithm G

Consider the following algorithm:

Algorithm G.

1. for $i = 2$ to $n$ /* process row $i$, $i$ from 2 to n */
   2. begin
   3. for $j = 1$ to $i-1$ /* part 1 */
   4. if $M(i,j)=1$ then
   5. $row(i) = row(i) \lor row(j)$;
   6. for $j = i-1$ down to 1 /* part 2 */
   7. if $M(j,i)=1$ then
   8. $row(j) = row(j) \lor row(i)$;
   9. end;

Algorithm G makes only one pass over the matrix. During part 1, it attempts to update row $i$ using rows $1, 2, \ldots, i-1$, while during part 2 it attempts to update rows $1, 2, \ldots, i-1$ using the recently updated row $i$. For each $i = 2, \ldots, n$, first (part 1) it scans from left to right the entries of row $i$ which are left of the main diagonal and secondly (part 2) it scans from bottom to top the entries of column $i$ which are above the main diagonal. Whenever an entry $M(a,b)$ is found to be equal to 1, the operation $row(a):=row(a) \lor row(b)$ is performed.

In the remaining part of this section we illustrate how algorithm G works through an example, and then we give a proof of its correctness.

Example for algorithm G.
Let $M$ be the input matrix as shown below.

$$
M = \begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

The path $P$: $4 \rightarrow 1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 6$ exists. Essentially what the algorithm does is that for any path such as $P$, there will be an edge from the beginning vertex (4 in our example) to the end vertex (6 in our example) after the processing of the highest numbered row (5 in our example) which corresponds to the same numbered node in the path, but excluding the end node.

Proof of correctness of algorithm G.
It is essential to show that algorithm G sets $M(a,b) = 1$ iff there exists a directed path from $a$ to $b$ in the input matrix. The "if" part of the result is established by Lemma 3.3 and the "only if" part is established by Lemma 3.2. Notation: Given a path $P$: $a_1 \rightarrow \ldots \rightarrow a_n$, we will refer to the max{$a_1, \ldots, a_{n-1}$} as the $\text{maxP}$. For example, $\text{maxP} = 5$ for the path $P$ in the above example.

---

2 In lines 4 and 8 of the algorithm, $\lor$ denotes the logical "or" operation applied to each element of the rows.