GRAPH TRAVERSAL WITH MINIMUM STACK DEPTH

Terry Beyer
Sandra Hedetniemi
Stephen Hedetniemi
Andrzej Proskurowski

Department of Computer and Information Science
University of Oregon
Eugene, OR 97403

Abstract

A vertex traversal of a graph is a sequence of open and close operations applied to the vertices of the graph subject to the following constraints. Any vertex may be opened initially. Thereafter, only a vertex which is adjacent to a currently open vertex may be opened. A vertex may be closed at any time but may never be reopened. A shift operation which simultaneously closes one vertex and opens an adjacent vertex is also allowed. The traversal is complete when all vertices have been opened and closed. The depth of a traversal is the maximum number of vertices open at any moment during the traversal. The depth of a vertex is the minimum depth of any traversal starting at that vertex. A vertex of minimum depth is called an optimal vertex.

We show the following: the problem of computing the depth of a vertex in an arbitrary graph is NP-complete; the depths of any two vertices in a tree differ by at most one; and an algorithm can be constructed for finding an optimal vertex in a tree and its depth, which is linear in the number of vertices in the tree. We also characterize the locations of optimal vertices in a tree.

1. Introduction

In most network theoretic computations it is necessary to examine every vertex or edge of a network in some order. A standard method for doing this is to employ a recursive procedure, due to Tarjan [3], called depth-first search. In this procedure one visits (examines) a vertex only if it has a previously visited, neighboring vertex. In the process of visiting a new vertex the previously visited, adjacent vertex is placed on a pushdown stack in order to remember how the vertex was reached. The process of visiting every vertex in this way creates a traversal of the network. It is natural to ask, among all depth-first traversals of a given network, what is the minimum number of vertices that must be placed on the pushdown stack?

This question has important practical consequences when performing depth-first traversals of very large networks, since the number of items on the pushdown stack determines a minimum amount of additional memory which is required to carry out the computation. In one case in point, a computation on a Galois field with 10^8 elements involved the construction of a coset graph. Each new vertex that was visited in this coset graph required an additional 10^3 elements to be placed on the pushdown stack.

In this paper we show that the problem of determining the minimum number of vertices that must be placed on the pushdown stack in a depth-first traversal of an arbitrary graph is NP-complete. However, we present a linear time solution to this problem when the graph is a tree.

In a somewhat related paper, Kemp [2] has considered the number of registers needed to evaluate a binary tree.

2. Definitions

A vertex is said to be <u>open</u> when it is placed on a pushdown stack; it is said to be <u>closed</u> when it is removed from the stack. A <u>traversal of a graph G from an initial vertex v</u> is a sequence of open and close operations on the vertices of G such that:

1) the first operation is to open v;

2) each vertex is opened and closed exactly once; and

3) a vertex may be opened only if

   a) it is adjacent to an open vertex, or

   b) it is adjacent to a vertex which has just been closed.

We refer to the operation in 3b) as the slide operation. The motivation for allowing a vertex to be opened if it is adjacent to a vertex which has just been closed comes from the coset application and corresponds to the process of transforming one coset into another by multiplying each element on the right by a fixed group element. This simultaneously closes the first coset and opens the second.

The (stack) depth of a traversal of a graph G is the maximum number of vertices open at any time during the traversal. The depth of a vertex v is the minimum depth of any traversal of G originating at v. An optimal vertex is a vertex of minimum depth in G.

3. Preliminary results

Proposition 1. Let G be an arbitrary graph. The problem of determining if G has a vertex of depth k, for a fixed $k \geq 1$, is NP-complete.

Proof. It is easy to see that a graph G has a vertex of depth 1 if and only if G has a Hamiltonian path (cf. [1, p.199]).

Although the problem of finding an optimal vertex in an arbitrary graph is NP-complete, when restricted to trees, it can be solved in linear time. In what follows we will restrict our attention to trees and demonstrate this result. In Section 4 we present a linear algorithm for finding an optimal vertex in an arbitrary tree T. The results in this section can be used to prove the correctness of this algorithm.

Note that given a traversal of an arbitrary graph G, we may extract a rooted, spanning tree from the traversal by taking its starting vertex as the root and by taking as the parent of any other vertex v, that adjacent vertex p(v), which was open (or just closed) at the time v was opened, under condition 3) in the definition of a traversal. If more than one vertex is a candidate for p(v), we arbitrarily choose the one most recently opened.

It can be proven, although it will be omitted here, that for any traversal of depth d, there is an equivalent traversal of depth d or less which is also a depth-first traversal of the corresponding spanning tree. The proof proceeds by showing how to reorder the sequence of open and close operations without increasing the depth of the traversal.

The problem of finding an optimal vertex in a tree which is a path is trivial. Each endvertex is optimal, of depth 1, and every other vertex is non-optimal, of depth 2. Therefore, we will consider only trees T which are not paths.

Let u and v be adjacent vertices in a tree T and let T_v^u be the maximal subtree of T containing v but not u (cf. Figure 1). The influence of v on u, denoted inf(v,u), equals the depth of v in T_v^u.

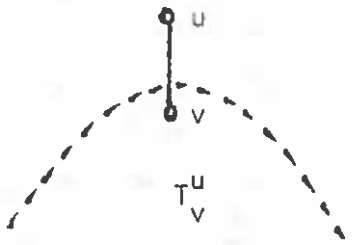

Figure 1.

The following result relates the depth of a vertex v to the maximum influence on v of its neighbors.

Theorem 2. Let v be a vertex in a tree T and let m be the maximum influence of any neighbor of v on v. Then the

$$\text{depth of } v \text{ in } T = \begin{cases} m & \text{if only one vertex adjacent to } v \\ & \text{has influence } m \text{ on } v \\ m+1 & \text{otherwise} \end{cases}$$

Proof. In any traversal of G from the initial vertex v we must hold v open while traversing each of the neighboring subtrees except the last, to which we may slide. Thus, if the maximum influence comes from a unique neighbor, we may slide to that neighbor last and its depth in its subtree will determine the depth of the optimal traversal from v.

However, if there are two or more neighbors exerting maximum influence on v then we must hold v open during the traversal of one of these subtrees and hence must add 1 to its influence. We assume of course that all subtrees are traversed in an optimal way.

Corollary 3. Let u and v be two adjacent vertices in a tree T and let inf(u,v) = d. Then every vertex in T_v^u has depth in T not less than d.

Proof. This follows easily from the definition of influence and Theorem 2.

Theorem 4. The depths of any two vertices in a tree T differ by at most one.

Proof. Let d be the maximum depth of any vertex in T and let v be a vertex of depth d. By Theorem 2 either v has two or more neighbors exerting influence d-1 on v or v has a unique neighbor exerting an influence of d. In the latter case, one can follow a chain of unique maximum influences back until one reaches a vertex w having two or more neighbors whose influence on w is d-1.

Now every vertex of T is under the propagated influence of at least one of these two vertices, and hence, by Corollary 3, can have depth no less than d-1.

A vertex of depth d is called critical if it has three or more neighbors whose influence on it is d-1 or greater.

Theorem 5. A critical vertex is optimal and exerts the same influence on each of its neighbors.

Proof. Let v be a critical vertex of depth d. Then v exerts an influence of d on each of its neighbors, since no matter which neighbor, say u, is chosen, v has at least two other neighbors whose influence of d-1 or more combines in v to exert an influence of d on u. Since v is of depth d it cannot exert an influence greater than d. The result now follows from Corollary 3.

Theorem 6. Any tree T has at most two critical vertices.

Proof. Assume to the contrary that a tree T has three critical vertices. By Theorem 5 they are all optimal and hence must all be of

the same depth d. If they all lie on a common path, then the critical vertex in the middle will receive a propagated influence of d from each of the other two critical vertices, and hence cannot be of depth d. If the three vertices do not lie on a path then the propagated influence of any two will combine at an intermediate vertex to form an influence of d+1, which will raise the depth of the third critical vertex to more than d. Thus, we have a contradiction in either case.

Theorem 7. Let v be a non-critical vertex of a tree T. Then v is non-optimal if and only if there exist two critical vertices either of which can be reached from v by a path not containing the other vertex.

Proof. Let d be the depth of an optimal vertex of T. If either of two critical vertices is reachable from v without passing through the other, then their influence of d will combine to raise the depth of v to d+1, making v non-optimal.

Now suppose that only one critical vertex, say y, can be reached from v without passing through another critical vertex. We must show that v has minimum depth d. Suppose not. By Theorem 4 v must have depth d+1. Without loss of generality we can assume that no vertex on the path from v to y has depth d+1. Assume that vertex u is the neighbor of v which is closer to (or is) the critical vertex y. Then $inf(u,v) = d$, for otherwise u would have depth d+1, which contradicts our assumption about v. Since the depth of v is d+1, there must exist a second neighbor of v, say w, with $inf(w,v) \geq d$. But $inf(w,v) < d+1$, for otherwise the influence of d+1 would propagate to the critical vertex whose depth is only d. Thus, $inf(w,v) = d$. Since v is not critical, all neighbors of v except u and w must have an influence of less than d on v. Thus, $inf(v,w) = d$.

We now focus our attention on the edge (v,w). It has these properties: (1) $inf(v,w) = inf(w,v) = d$, and (2) there is no critical vertex in T_w^v. Now choose an edge (v',w') having these two properties, which is farthest from the critical vertex y reachable from v. Now $inf(w',v') = d$ and since w' is not critical, w' must have one other neighbor x' such that $inf(x',w') = d$. Furthermore, all other neighbors of w' must have influence $< d-1$ on w'. But then we know that $inf(w',x') = d$. Since there is no critical vertex in $T_{w'}^{v'}$, there is no

critical vertex in $T_{x'}^{w'}$. Hence, we have found an edge satisfying the two properties above, which is farther from a critical vertex than (v',w'), which contradicts our selection of (v',w') as the farthest such edge.

4. An algorithm for finding an optimal vertex in an arbitrary tree

We next describe an algorithm which prunes an arbitrary tree T, one leaf at a time, reducing it to a single vertex. We then prove that this vertex is critical in T. During the pruning process the reduced tree consists of the set of vertices of T not yet pruned. The influence of a leaf is its influence on its unique neighbor. We associate with each vertex v enough information so that when v is pruned, we can compute its influence. For this purpose it is sufficient to associate two values, max and mult, with each vertex: for an interior vertex v, max(v) is the maximum influence on v of all neighbors of v which have already been pruned, and mult(v) is the number of such maximum neighbors; for each leaf of T, max and mult are initialized to 1.

Pruning Algorithm [To reduce an arbitrary, non-path tree T to a critical vertex]

[Initialize]

For every interior vertex u of T, set max(u) = mult(u) = 0.

For every leaf v of T, set max(v) = mult(v) = 1.

While T contains more than one vertex do

[Select a leaf]

If there was an immediately preceding pruning step and that step created in the reduced tree a new leaf whose influence equals the minimum influence of any leaf in the reduced tree

then select that leaf

else select any leaf of minimum influence in the reduced tree

end if

[Compute the influence of the leaf]

Let the leaf selected be v and its unique neighbor in the reduced tree be u.

Set inf(v,u) = (if mult(v) = 1 then max(v) else max(v)+1)

```
    [Update values for u]
      if inf(v,u) > max(u)
        then max(u) = inf(v,u)
             mult(u) = 1
        else if inf(v,u) = max(u)
               then mult(u) = mult(u)+1
               end if
        end if
    [Prune the leaf]
      Set T = T - v
  end while
end algorithm
```

We observe, without proof, the following:

(1) The vertices of T are pruned in non-decreasing order of their influence. This follows from the selection criteria employed by the algorithm and the propagation of influence.

(2) At any given moment the leaves of the reduced tree have at most two values of inf. This is because of (1) above and Theorem 4.

(3) The final vertex in T has depth > 1 and vertex degree ≥ 3. This follows from (1), the fact that T is a non-path tree, and the fact that the selection criteria prunes all vertices of degree 2 in T as soon as they become leaves in the reduced tree.

(4) The algorithm can be implemented so that it executes in linear time in the number of vertices of T. A double-ended queue can be used to maintain the leaves in the reduced tree with all leaves of minimal inf preceding all vertices of non-minimal influence.

Theorem 8. The vertex which remains unpruned at the termination of the Pruning Algorithm is critical in T.

Proof. Let v be the vertex which remains at the end of the Pruning Algorithm and let its depth in T be d. Let k be the degree of v in T. By observation (3), we know that $k \geq 3$. Thus, $d \geq 2$. Let

the neighbors of v in T be $u_1, u_2, \ldots, u_k$, where u_i is pruned before u_{i+1}, for all i. By observation (1), we know that $\inf(u_i,v) \leq \inf(u_{i+1},v)$, for all i, $1 \leq i \leq k-1$.

Now assume that v is not critical. Then $\inf(u_j,v) \leq d-2$, for $j \leq k-2$. Since the depth of v is d we must have either $\inf(u_{k-1},v) = \inf(u_k,v) = d-1$ or $\inf(u_{k-1},v) \leq d-1$ and $\inf(u_k,v) = d$. We will now show that in either case v must be pruned before u_k, due to the selection criteria. In the first case, v must be pruned immediately after u_{k-1}, since $\inf(u_{k-1},v) = d-1 = \inf(v,u_k)$ implies that the Pruning Algorithm is currently pruning leaves with influence d-1 and v is the newest such leaf. In the second case, after u_{k-1} is pruned we will have $\inf(v,u_k) < d = \inf(u_k,v)$, and hence v must be pruned before u_k. Thus, v must be critical.

5. Summary

Combining Theorems 4 through 8, we see that every non-path tree T has either one or two critical vertices. All vertices are optimal if T has only one critical vertex and if T has two critical vertices, then the non-optimal vertices have depth only one greater than optimal and lie in that portion of T which is reachable from either critical vertex without passing through the other.

Finally, the Pruning Algorithm can easily be extended so that

(1) the depth of every vertex in T can also be computed, by backtracking, and

(2) both critical vertices can be found if two of them exist.

Since the problem of finding the depth of a vertex in an arbitrary graph is NP-complete, while in an arbitrary tree it can be found in linear time, it would be interesting to find other classes of graphs for which the depth can be computed in polynomial time.

6. Bibliography

[1] Garey, M. R. and Johnson, D. S., Computers and Intractability: a guide to the theory of NP-completeness, W. H. Freeman and Co., San Francisco, 1979.

[2] Kemp, R., The average number of registers needed to evaluate a binary tree optimally, Acta Informatica, 11(1979), 363-372.

[3] Tarjan, R., Depth-first search and linear graph algorithms, SIAM J. Comput., 1 (1972), 146-160.