CIS-TR-80-16

VERTEX AND EDGE DELETION ALGORITHMS FOR TREES*

by

Sandra Mitchell Hedetniemi**

Stephen T. Hedetniemi**

Abstract

In KD[79] Krishnamoorthy and Deo presented some 17 vertex-deletion problems which are NP-complete for arbitrary graphs. In addition they showed that many of these problems remain NP-complete when restricted to either planar graphs or bipartite graphs. This paper is an attempt to complete one aspect of their study by considering the algorithmic complexity of the same problems when restricted to trees.

We show that most of the corresponding edge deletion problems for trees have already been solved algorithmically, each in linear time, yet apparently none of the vertex deletion problems have been considered. We present linear time solutions to most of these problems.

* Research supported in part by the National Science Foundation under Grant MCS-79-03913.

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

1. Introduction

Given an arbitrary graph G = (V,E) the general vertex- (edge-) deletion problem can be stated as follows: determine the minimum number of vertices (edges) that can be removed from G so that the remaining graph has property Π. Typical properties Π that one might consider are that a graph be (i) planar, (ii) acyclic, (iii) bipartite, (iv) complete, (v) Hamiltonian, or (vi) Eulerian.

The vertex-deletion problem can also be stated as a maximization problem as follows: determine the maximum number of vertices in a set S such that the subgraph induced by S has property Π.

Unfortunately, for most of the interesting properties Π that one might want to consider, the corresponding vertex-deletion problems tend to be NP-complete. For example, in KD[79] Krishnamoorthy and Deo present some 17 examples of vertex-deletion problems which are NP-complete. Other examples can also be found in Garey and Johnson's book GJ[79]. Krishnamoorthy and Deo also point out that many of the same vertex-deletion problems remain NP-complete when the arbitrary graphs are restricted to be either planar or bipartite.

The purpose of this paper is to see what happens when these vertex- (edge-) deletion problems are further restricted to trees. We will show that most of the corresponding edge-deletion problems for trees have already been solved, although they are stated in the form of vertex-partition problems. However, it appears that the vertex-deletion problems for trees have not been studied. We present linear algorithms for solving most of these problems.

2. Edge-deletion problems for trees

It is easy to see that when restricted to trees the general edge-deletion problem is equivalent to the following general vertex-partition problem: partition the vertices V of an arbitrary tree T into a minimum number of

subtrees each of which has property Π. In particular, the edges of T which join vertices in different subtrees of such a partition are those whose deletion produces the desired partition.

Although the properties Π of being planar, bipartite, Eulerian or acyclic have no particular interest in the context of trees, the following properties are of some interest:

(i) is a path;

(ii) is a star;

(iii) has diameter $\leq k$;

(iv) has maximum degree (Δ) $\leq k$;

(v) is a complete graph, i.e. consists of a single vertex (K_1) or two adjacent vertices (K_2);

(vi) is a path of length $\leq k$; and

(vii) is a star with $\leq k$ vertices.

As indicated in Table 1 algorithms for solving the edge-deletion problem for the first six of these properties have already been constructed.

The b-matching algorithm for trees in GHT[76] solves problems 5, 1 and 4 above for the b-values of 1, 2 and k, respectively. The algorithm for the domination number of a tree in CGH[75] solves problem 2 above. Kariv and Hakimi KH[80] have an algorithm which can be used to solve problem 3 above, although Farley, Hedetniemi and Proskurowski FHP[80] independently obtained an algorithm for solving this problem directly. Any algorithm for problem 3, incidentally, also solves problem 2 when k = 2.

Finally, in HH[79] an algorithm is given for decomposing an arbitrary tree into a minimum number of paths, each of which has length $\leq k$. All of the above mentioned algorithms are linear in that they require O(n) time for a tree with n vertices.

To the best of our knowledge, an algorithm for solving the edge-deletion problem for property 7 above has not been constructed, although one can be constructed by modifying the vertex-deletion algorithm for property 7 in this paper.

3. Vertex-deletion algorithms for trees

We conclude this paper by presenting solutions to each of the above-mentioned vertex-deletion problems for trees, as indicated in Table 1.

Problem 4: Delete a minimum number of vertices from a tree so that each of the remaining subtrees has maximum degree $\leq$ K.

The following two lemmas provide the basis for a simple algorithm for solving this problem.

Lemma 1. Let u be the root of a subtree T_u of a tree T (possibly $T = T_u$), and let the degree of every other vertex in T_u be $\leq$ K (cf. Figure 2). Then there exists a minimum set of vertices S containing vertex u, whose deletion from T results in a collection of subtrees, each of which has maximum degree $\leq$ K.

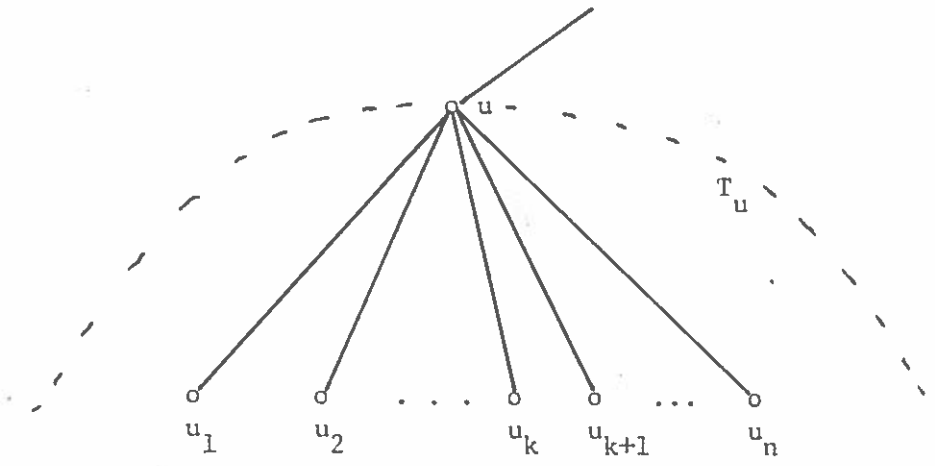

Figure 2

Lemma 2. Let u be a vertex in a tree T which is adjacent to a vertex v, let T_v be the subtree of T - (u,v) rooted at v, let the degree of v in T_v be K and assume that the degree of every other vertex in T_v is $\leq$ K (cf. Figure 3). Then there exists a minimum set of vertices S containing vertex u whose deletion from T results in a collection of subtrees, each of which has maximum degree $\leq$ K.

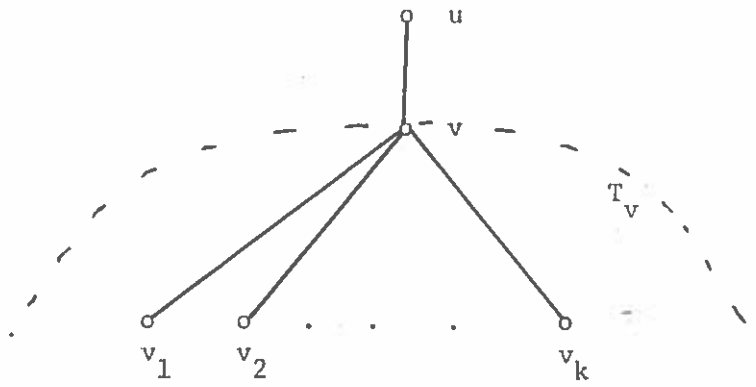

Figure 3

Algorithm SUBTREE. To find a minimum set of vertices whose deletion from a rooted tree T results in a collection of sub-trees, each of which has maximum degree $\leq$ K; the rooted tree T with N vertices is represented by the parent array TREE[1..N]; the array DEG[1..N] is used to describe the final disposition of each vertex as follows: DEG(I) = K+1 means that vertex I is to be deleted; otherwise DEG(I) = J means that the current degree of vertex I is J.

```
[Initialize all vertices to zero]
    for I ← 1 to N do
        DEG(I) ← 0
        od
```

```
[Process each vertex from N to 2]
    for V ← N downto 2 do
        [get the parent of vertex V]
        U ← TREE(V)
        [remove vertex U if DEG(V) = K]
        case  DEG(V):
              0 ≤ DEG(V) ≤ K-1 :  if DEG(U) ≤ K  then DEG(U) ← DEG(U) + 1 fi
                        K      :  DEG(U) ← K + 1
                    ≥ K + 1    :  [do nothing, vertex V is to be removed]
              endcase
        od.
[At this point vertex 1 will already have been processed]
```

It is easy to see that Algorithm SUBTREES not only solves Problem 4, but by setting K = 1 it solves Problem 5 and by setting K = 2 it solves Problem 1.

Problem 3: Delete a minimum number of vertices from a tree so that each of the remaining subtrees has diameter $\leq$ K.

The correctness of Algorithm SUBDIAMETER for solving this problem is based on the following simple lemma.

Lemma 3. Let u be the root of a subtree T_u of a rooted tree T (possibly $T_u = T$) and let the descendants of u in T_u be $u_1, u_2, \ldots, u_m$. Furthermore, let diam $(T_u) > K$ and let diam $(T_{u_i}) \leq K$ for $1 \leq i \leq m$. Then there exists a minimum set of vertices S containing u, whose deletion from T results in a collection of subtrees, each of which has diameter $\leq$ K.

Proof. Let S be a minimum set of vertices whose deletion from T results in subtrees of diameter $\leq$ K and assume that $u \notin S$. Consider $S \cap T_u$. Clearly,

$|S \cap T_u| \geq 1$, since diam $(T_u) > K$. But then $S' = S - S \cap T_u \cup \{u\}$ is a minimum set of vertices containing vertex u whose deletion from T results in subtrees of diameter $\leq K$.

Algorithm SUBDIAMETER. To delete a minimum number of vertices from a tree so that each remaining subtree has diameter $\leq K$; a tree with N vertices is represented by a parent array TREE [1..N]; the array LONG [1..N] records for each vertex v, current value of the length of a longest path from v to an endvertex in the subtree T_v rooted at v; the array DIAM [1..N] records for each vertex v, the current value of the diameter of the subtree T_v; when the algorithm is finished all vertices v with a value DIAM(v) > K are to be deleted.

```
[Initialize diameters and longest paths to zero]
    for I ← 1 to N do
        DIAM(I) ← 0
        LONG(I) ← 0
        od
[Process each vertex from N to 2]
    for V ← N downto 2 do
        [get parent of v]
        U ← TREE(V)
        [update diameters and longest paths]
        if DIAM(U) ≤ K and DIAM(V) ≤ K
            then DIAM(U) ← max {DIAM(U), DIAM(V), LONG(U) + LONG(V) + 1}
                 LONG(U) ← max {LONG(U), LONG(V) + 1}
            fi
        od.
[At this point vertex 1 has been processed]
```

In addition to solving Problem 3, Algorithm SUBDIAMETER also solves Problem 2 when the value of diameter K = 2.

Problem 6: Delete a minimum number of vertices from a tree so that each of the remaining subtrees is a path of length $\leq$ K.

Rather than stating a lemma concerning the correctness of the following algorithm (which can easily be done); we will instead describe in more detail how it works.

With each vertex v we associate a variable STATE(v) having one of four possible values:

STATE(v) = 0 means that vertex v has not yet been considered;

STATE(V) = 1 means that vertex v has one path attached to it of length $\leq$ K; the length of this path is given by the variable LENGTH(v);

STATE(v) = 2 means that vertex v has two paths attached to it, the sum of whose lengths is $\leq$ K;

STATE(v) = 3 means that vertex v is to be deleted, in order to produce subtrees which are paths of length $\leq$ K.

Using the parent array representation of a tree, we proceed right-to-left across the array examining in turn the state of each vertex v together with the state of its parent vertex u = TREE(v). As in Algorithm SUBTREES, Algorithm SUBPATHS is simply an iteration containing one CASE statement, depending on the pair of values (STATE(v), STATE(u)).

Algorithm SUBPATHS. To delete a minimum number of vertices from a tree so that each remaining subtree is a path of length $\leq$ K; the tree is represented by a parent array TREE [1..N]; the array STATE [1..N] is used to describe the

final disposition of each vertex, where STATE(u) = 3 means that vertex u is to be deleted.

```
[Initialize each vertex]
    for I ← 1 to N do STATE(I) ← 0 od
[Process each vertex]
    for V ← N downto 2 do
        [get the parent of vertex v]
        U ← TREE(V)
        [apply the case statement]
        case ( STATE(V), STATE(U) ):
             (0,0): STATE(U) ← 1     [U now has one attached path]
                    LENGTH(U) ← 1
             (0,1): if LENGTH(U) = K
                        then STATE(U) ← 3    [delete U]
                        else STATE(U) ← 2    [U now has two attached paths]
                        fi
             (0,2): STATE(U) ← 3     [delete U]
             (1,0): if LENGTH(V) = K
                        then STATE(U) ← 3    [delete U]
                        else STATE(U) ← 1    [U has one attached path]
                             LENGTH(U) ← LENGTH(V) + 1
                        fi
             (1,1): if LENGTH(U) + LENGTH(V) + 1 > K
                        then STATE(U) ← 3    [delete U]
                        else STATE(U) ← 2    [U has two attached paths]
                        fi
             (1,2): STATE(U) ← 3     [delete U]
```

```
        (2,x):  STATE(U) ← 3     [delete U]
(Otherwise ):   [do nothing]
     endcase
  od.
```

Problem 7. Delete a minimum number of vertices from a tree so that each of the remaining subtrees is a star with $\leq$ K vertices.

As in Algorithm SUBPATHS, an Algorithm SUBSTARS can be constructed for solving Problem 7 which is a simple iteration containing a single CASE-statement. We associate with each vertex v a variable STATE(v) having one of five possible values:

STATE(v) = 0 means that vertex v has not been encountered yet;

STATE(v) = 1 means that vertex v has a single vertex attached to it;

STATE(v) = 2 means that vertex v has $\geq$ 2 vertices attached to it; in this case the variable SIZE(v) records the number of vertices attached to it;

STATE(v) = 3 means that vertex v is to be deleted in order to leave stars with $\leq$ K vertices;

STATE(v) = 4 means that vertex v is an endvertex of a star whose central vertex is a descendent of v.

In the interest of brevity, we only provide the corresponding CASE-statement for this algorithm; the rest of the algorithm is identical to that in Algorithm SUBPATHS.

CASE (STATE(V), STATE(U)):

STATE(V): \ STATE(U):	0	1	2	3	4
0	STATE(U) ← 1 SIZE(U) ← 1	STATE(U) ← 2 SIZE(U) ← 2	if SIZE(U) = K then STATE(U) ← 3 else SIZE(U) ← SIZE(U)+1 fi	[do nothing]	STATE(U) ← 3
1	STATE(U) ← 4	STATE(U) ← 3	STATE(U) ← 3	[do nothing]	STATE(U) ← 3
2	if SIZE(V) = K then STATE(U) ← 3 else STATE(U) ← 4 fi	STATE(U) ← 3	STATE(U) ← 3	[do nothing]	STATE(U) ← 3
3	[do nothing]	[do nothing]	[do nothing]	[do nothing]	[do nothing]
4	STATE(U) ← 3	STATE(U) ← 3	STATE(U) ← 3	[do nothing]	STATE(U) ← 3

ENDCASE.

One final point concerns the construction of an edge-deletion algorithm which leaves stars with K vertices. Such an algorithm can be constructed which is very similar to Algorithm SUBSTARS. It too contains a CASE-statement with 16 possible pairs of values for (STATE(V), STATE(U)). We leave the details to the interested reader.

4. Summary

We have shown that most vertex and edge deletion problems can be solved algorithmically in linear time for trees, althougn for bipartite or planar graphs Krishnamoorthy and Deo KD[79] have shown they are usually NP-complete.

An interesting by-product of this study is the realization that most of these algorithms for trees, using parent arrays, can be described as a simple iteration containing a single CASE-statement, which compares a finite-state value of a vertex with that of its parent vertex. It would be interesting to know how widely this CASE-statement approach can be applied to tree algorithms in general.

5. Bibliography

CGH[75] Cockayne, E., Goodman, S. and Hedetniemi, S. T., A linear algorithm for the domination number of a tree, Information Processing Lett., 4 (1974), 41-44.

FHP[80] Farley, A., Hedetniemi, S. T. and Proskurowski, A., Partitioning trees: matching, domination and diameter $\leq$ k, Univ. of Oregon Tech. Rept. CIS-TR-80-2, 1980.

GHT[76] Goodman, S., Hedetniemi, S.T. and Tarjan, R., B-matchings in trees, SIAM J. Comput., 5 (1976), 104-107.

GJ[79] Garey, M. and Johnson, D., Computers and Intractability: A Guide to NP-completeness, Freeman and Co., San Francisco, 1979.

KD[79] Khrishnamoorthy, M. and Deo, N., Node-deletion NP-complete problems, SIAM J. Comput., 8 (1979), 619-625.

KH[79] Kariv, O. and Hakimi, S., An algorithmic approach to network problems I; the p-centers, SIAM J. Appl. Math., 37 (1979),513-538.

HH[79] Hedetniemi, S. M. and Hedetniemi, S. T., Broadcasting by decomposing trees into paths of bounded length, Univ. of Oregon Tech. Rept. CS-TR-79-16, 1979.

Property Π	Remove a minimum number of edges from a tree so that each remaining subtree has property Π	Remove a minimum number of vertices from a tree so that each remaining subtree has property Π
1. is a path	GHT[76]	< this paper >
2. is a star	CGH[75]	< this paper >
3. has diameter $\leq k$	KH[80], FHP[80]	< this paper >
4. has $\Delta \leq k$	GHT[76]	< this paper >
5. is a K_1 or a K_2	GHT[76]	< this paper >
6. is a path, length $\leq k$	HH[79]	< this paper >
7. is a star, $\leq k$ vertices	< this paper >	< this paper >
Problem	(a)	(b)

Table 1.