CIS-TR-80-2

PARTITIONING TREES: MATCHING, DOMINATION AND MAXIMUM DIAMETER

by

Arthur Farley*

Stephen Hedetniemi*+

Andrzej Proskurowski*

* Research supported in part by the National Science Foundation under Grant STI 7902960

+ 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

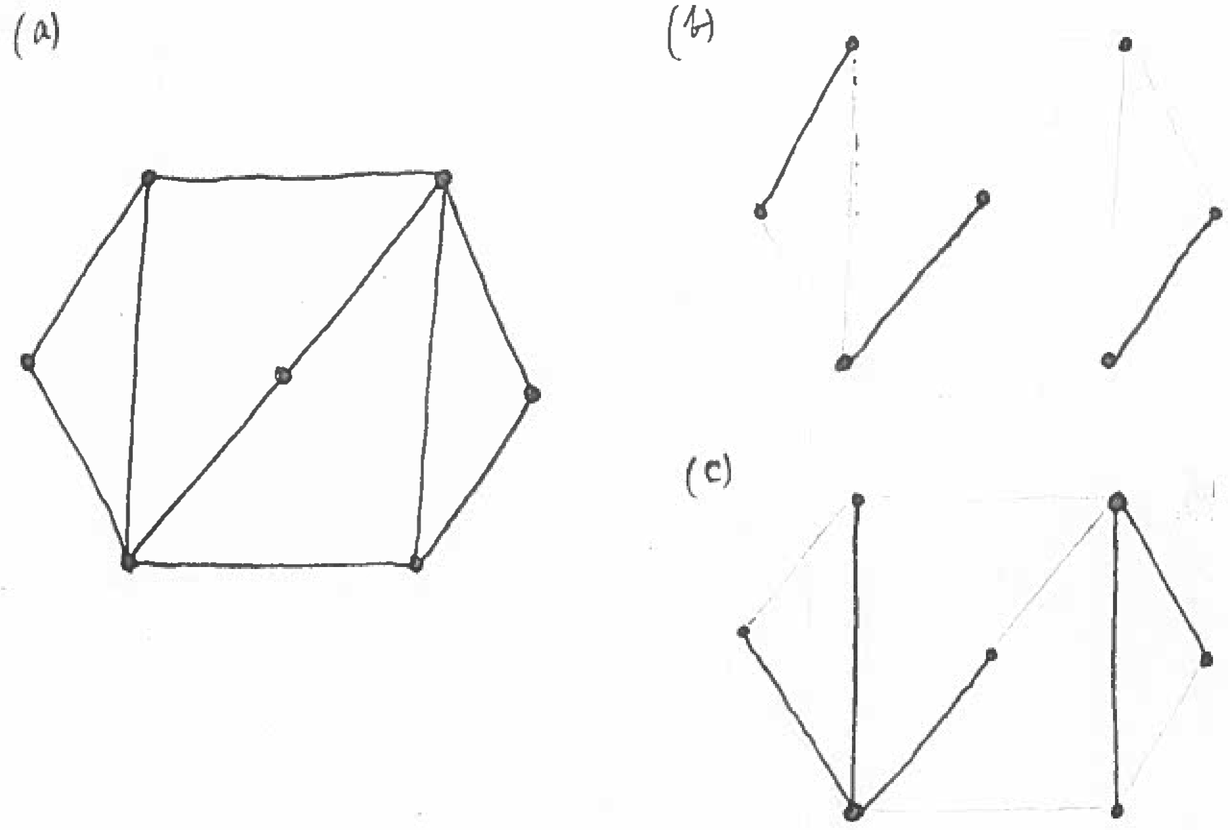
1. Introduction

A matching in a graph $G=(V,E)$ is a collection M of edges of G such that no two edges have a vertex in common. A maximum matching in G is a largest matching in G. A dominating set in a graph G is a set D of vertices such that every vertex not in D is adjacent to at least one vertex in D. A minimum dominating set in G is a smallest dominating set in G.

A subtree partition of a graph $G=(V,E)$ is a set of vertex disjoint, acyclic subgraphs (trees) of G: $T_1=(V_1,E_1),\ldots,T_p=(V_p,E_p)$, such that $\bigcup V_k=V$, $V_i \cap V_j=\emptyset$ and $E_k \subseteq E$ for all $i \neq j,k$; $1 \leq i,j,k \leq p$. The diameter of a graph is the maximum distance between a pair of vertices, where the distance between a pair of vertices is the fewest number of edges on a path connecting them. One basis for a subtree partition of a graph would be to require all subtrees to have diameter less than some prespecified bound. An application of diameter-bounded partitions would be the determination of communication (or transportation) subnetworks with limited maximum separation of members.

A matching M and a dominating set D in a graph G are related in that they determine diameter-bounded subtree partitions of G. The subgraph $G_1=(V,M)$ constitutes a partition of G into subtrees of diameter at most 1. The subgraph $G_2=\langle V,E'\rangle$, where $E' \subseteq E$ is such that every vertex not in D is connected to exactly one vertex in D, constitutes a partition of G into subtrees of diameter at most 2. Conversely, any subtree partition of G having maximum diameter less than or equal to 1 or 2 determines a matching or a dominating set, respectively.

For a maximum matching M_{max} and a minimum dominating set D_{min}, the associated partitions have the fewest numbers of trees. Figure 1 presents an example of a graph G, a maximal matching and a minimum dominating set with an associated subtree partition of G.



<u>Figure 1</u> (a) Graph G, (b) a maximum matching, and (c) subtree partition defined by a minimum dominating set (bold vertices).

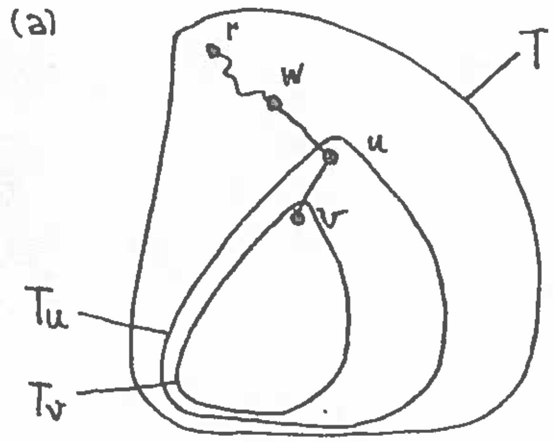
The computational complexities of determining M_{max} and D_{min} in an arbitrary graph G appear to be different. Polynomial algorithms are known to exist for finding an M_{max} in an arbitrary graph (c.f. [2], [6]). However, the problem of determining a D_{min} in an arbitrary graph has been shown to be NP-complete [3]. Many graph theoretic problems currently unsolvable in an efficient manner for general graphs have efficient solutions for restricted classes of graphs. For trees, both M_{max} and D_{min} can be determined in linear time (c.f. [1], [4]). In this paper, we present a linear algorithm for partitioning an arbitrary tree into a minimum number of subtrees each having diameter at most k for any given k. By the correspondence between M_{max} and D_{min} and minimum-order, diameter-bounded, subtree partitions, our algorithm represents a generalization of the previously published algorithms for M_{max} and D_{min} in trees. The correctness of our algorithm is a direct consequence of the lemmas established in the next section.

2. Key lemmas

Let d(T) denote the diameter of a tree T and $p_k(T)$ denote the minimum order of a partition of T into subtrees of diameter at most k.

In what follows, we will consider a tree T rooted at a given, arbitrary vertex r. Let u be a vertex of T other than r and let w be the vertex adjacent to u which lies on the unique path in T from u to r (possibly, w=r). Let T_u denote the (connected) subtree of T rooted at u and not containing w. The situation is illustrated in Figure 2(a). We define the height of T_u in T, denoted $h(T_u)$, as the length of a longest path in T_u from u to a leaf vertex of T_u.

By the definition, if a vertex u is a leaf of T, then $h(T_u)=0$ and $d(T_u)=0\leq k$. For a non-leaf vertex u of T, there exists a vertex v in T_u adjacent to u, as shown in Figure 2(a).



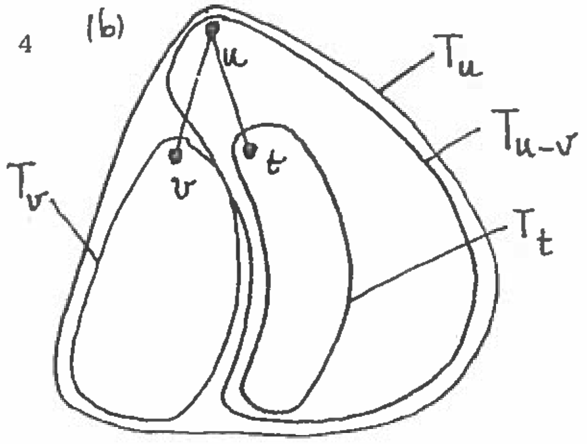


Figure 2 (a) A tree T rooted at r and (b) its subtree T_u.

Lemma 1 Let vertex u, $u\neq r$, have degree 2 in T.

Let v be as defined above, (see Figure 2(a)), such that $h(T_v)$, $d(T_v)\leq k$. If $h(T_v)=k$ then $p_k(T)=1+p_k(T-T_v)$, otherwise $h(T_u)\leq k$ and $d(T_u)\leq k$.

Proof If $h(T_v)=k$, then T_u cannot be an element of a subtree partition of G of diameter$\leq k$, (i.e., $p_k(T_u)=2$). On the other hand, $p_k(T_v)=1$. In $T-T_v$, vertex u is a leaf and as such has the minimum possible height. Therefore $p_k(T)=1+p_k(T-T_v)$. If $h(T_v)<k$ then $h(T_u)=1+h(T_v)\leq k$, and $d(T_u)=\max(1+h(T_v), d(T_v))\leq k$. □

In the case of vertex u of degree greater than 2, let T_{u-v} denote the tree T_u-T_v rooted at u. Let t be a vertex adjacent to u in T_{u-v} maximizing $h(T_t)$ over all subtrees T_t of T_{u-v}. As such, $h(T_{u-v})=h(T_t)+1$. Figure 2(b) illustrates this case's situation.

Lemma 2 Let vertex u have, $u\neq r$, degree greater than 2 in T. Let v and t be as defined above, and such that $h(T_{u-v})$, $h(T_v)$, $d(T_{u-v})$, $d(T_v)$, $d(T_t)\leq k$. Then:

(i) if $h(T_v)+1+h(T_{u-v})\leq k$

then $h(T_u)\leq k$, and $d(T_u)\leq k$;

(ii) if $h(T_v)+1+h(T_{u-v})>k$ and $h(T_v)>h(T_t)$

then $p_k(T)=1+p_k(T-T_v)$ and, in $T-T_v$, $h(T_u)\leq k$ and $d(T_u)\leq k$;

(iii) if $h(T_v)+1+h(T_{u-v})>k$ and $h(T_v)\leq h(T_t)$

then $p_k(T)=1+p_k(T-T_t)$ and, in $T-T_t$, $h(T_u)\leq k$ and $d(T_u)\leq k$.

Proof When $h(T_v)+1+h(T_{u-v})\leq k$, we have $h(T_u)=\max(h(T_v)+1, h(T_{u-v}))\leq k$ and $d(T_u)=\max(d(T_v), d(T_{u-v}), h(T_v)+1+h(T_{u-v}))\leq k$. If $h(T_v)+1+h(T_{u-v})>k$, then vertices v and t have to belong to different partitions of T, unless an edge of T_v or T_t is "cut", reducing the height of T_t or T_v. As such an action would introduce another component in the partition of G, the cut edge can best be either (u,v) or (u,t). If $h(T_v)>h(T_t)$, as in case (ii), then cutting the edge (u,v) results in two components: T_v and $T-T_v$. The subtree $T-T_v$ has a subtree partitioning of potentially smaller order than $T-T_t$ which would result from cutting the edge (u,t). In $T-T_v$, T_u is reduced to T_{u-v}, and thus $h(T_u)=h(T_{u-v})\leq k$ and $d(T_u)=d(T_{u-v})\leq k$. Because $d(T_v)\leq k$, $p_k(T)=1+p_k(T_{u-v})$. Similar argument proves the lemma in case (iii). □

3. Algorithm SUBTREES

The Algorithm SUBTREES below uses a recursive representation of a tree T. A recursive representation is based on a labeling of vertices 1,..., n, where for every non-root vertex i there is a unique vertex j adjacent to it, called its father, such that j<i. The pruning of the leaves is done by a linear scan of an array FATHER representing T. In FATHER, array entries correspond to the father vertices: FATHER [i]=j. During execution of the Algorithm SUBTREES, the edges incident to vertices of T which have been pruned are called processed, and the tree of not yet pruned vertices is called current. Processed edges may be cut (and placed in set C) or uncut, as determined by the algorithm. With each vertex v of T we associate two values, HEIGHT[v], and TALL[v]. They indicate the height of v and of the tallest pruned, still connected neighbor of v, in their respective, processed subtrees (i.e., $h(T_v)$ and $h(T_{TALL[v]})$).

Algorithm SUBTREES

Input: A tree $T=(V,E)$ with $|V|=n$, given by the array FATHER; and an integer k.

Output: Minimum size set of cut edges C, such that remaining subtrees have diameter $\leq k$.

Method:
```
[0.  initialize] for v:=1 to n do HEIGHT[v]:=0;
[1.  prune T] for v:=n downto 2 do
                    begin u:=FATHER[v]; h:=HEIGHT[v]+1;
                      if h+HEIGHT[u]>k
    [1.1 introduce cut]  then if h>HEIGHT[u]
                                then C:=C+(u,v)
                                else begin C:=C+(u,TALL[u]);
                                  HEIGHT[u]:=h; TALL[u]:=v end
    [1.2 no cut       ]  else if h>HEIGHT[u]
                               then begin HEIGHT[u]:=h;
                                  TALL[u]:=v end
                    end.
```

Theorem The algorithm SUBTREES computes the minimum order partitioning of the given tree T into subtrees of diameter at most k. The algorithm requires time and space directly proportional to the size of T.

Proof By Lemmas 1 and 2, the graph $P=(T,E-C)$ represents a partitioning of T into subtrees of diameter at most k. The order of this partition is $|C|+1$. In the algorithm, the invariant of the loop [1.] is $p_k(S)+|C|=$ constant, where S is the current tree extended by the connected (uncut) processed edges. To prove the invariant relation does hold throughout the execution of the algorithm, we notice that C changes only in Step 1.1, which is also the only place where S changes. The changes are such that $S':=S-S_z$ with

$p_k(S)=1+p_k(S')$ (Lemma 2, cases (ii) or (iii)) and $|C'|=|C|+1$, where $z\varepsilon\{v,TALL[u]\}$. Thus $p_k(S')+|C'|=p_k(S)-1+|C|+1=p_k(S)+|C|=$ const. Originally S=T and C=O so that the constant is $p_k(T)$. The correct height and tall values are maintained in 1.1 and 1.2. Upon exit from the loop, $d(S)\leq k$, $p_k(S)=1$ and then $p_k(T)=1+|C|$. Both steps [0.] and [1.] require only a constant total time to process each vertex. The space used by the algorithm consists of three vectors, each of length n, a set C of size at most n-1, and several scalar variables.

Algorithm SUBTREES can be straightforwardly generalized to the same problem with weighted edges and vertices. No change in the basic logic of the algorithm would be required.

Knowledge of the cut set of edges partitioning a tree T into subtrees of diameter $\leq k$ allows linear time determination of minimum absolute dominating set of radius k/2 in tree T by finding centers of the subtrees. The center of a given subtree can be found in time linear in the number of edges in the subtree [5]. A linear time algorithm has recently been published for determining a dominating set of arbitrary radius in a tree T [7]. From an absolute dominating set of radius k/2 one can determine a partition of T into subtrees of diameter $\leq k$ in time linear in the number of edges in T by a parallel, breadth first search from each element of the dominating set. To find the subtree corresponding to a single element of the dominating set would still require time linearly related to the number of edges in the original tree.

4. Conclusion

In this paper we have characterized M_{max} and D_{min} in a graph G as determining partitions of G into the fewest subtrees of diameter less than or equal to 1 and

2, respectively. We then present a general algorithm which determines a minimum order partition of an arbitrary tree into subtrees of diameter$\leq$k. The algorithm requires only linear time and space.

A linear algorithm which partitions trees into the fewest subtrees having less than or equal to k vertices has been previously published [6]. This vertex-based partitioning strategy also shares a direct correspondence with matching for number of vertices equal to two. It represents a different generalization of matching than the one we present, as trees with diameter less than or equal to two are not equivalent to trees having three or fewer vertices. We are investigating yet another generalization of matching, that of partitioning a tree into subtrees having maximum vertex degree less than or equal to k. These partitions correspond to matching for a maximum degree of one and path decomposition for maximum degree of two. A linear algorithm solving this problem for an arbitrary tree and integer k has been determined and will appear elsewhere.

5. Bibliography

[1] E.J. Cockayne, S.E. Goodman and S.T. Hedetniemi, A linear algorithm for the domination number of a tree, Information Processing Lett. 4 (1975), 41-44.

[2] S. Even and O. Kariv, An O($n^{2.5}$) algorithm for maximum matching in graphs, Proc. 16th Symp. on Foundation of Computing (1975), 382-399.

[3] M.R. Garey and D.S. Johnson, Computers and Interactibility: A Guide to the Theory of NP-completeness, W.H. Freeman, San Francisco, 1979.

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

[5] S. Halfin, On finding the absolute & vertex centers of a tree with distances, Transportation Sci., 8 (1974), 75-77.

[6] A. Itai, M. Rodeh and S. Tanimoto, Some matching problems for bipartite graphs, J. Assoc. Comput. Mach. 25 (1978), 517-525.

[7] O. Kariv and S.L. Hakimi, An algorithmic approach to network location problems I: the p-centers, SIAM J. on Ap. Math., 37 (1979), 513-538.

[8] S. Kundru and J. Misra, A linear tree partitioning algorithm, SIAM J. Computing, 6 (1977), 151-154.