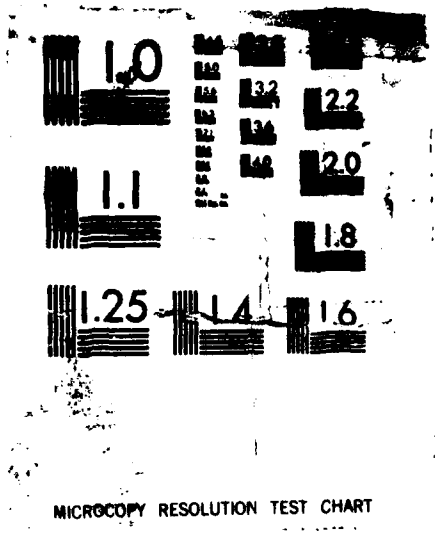
AD-A182 315 ON A NODE RANKING PROBLEM OF TREES AND GRAPHS(U) 1/1
GEORGIA INST OF TECH ATLANTA PRODUCTION AND
DISTRIBUTION RESEARCH CENTER A Y IYER ET AL. 24 OCT 86
UNCLASSIFIED PDRC-87-03 N00014-86-K-0173 F/G 12/4 NL



MICROCOPY RESOLUTION TEST CHART

ON A NODE RANKING PROBLEM OF TREES AND GRAPHS

Ananth. V. Iyer
H. Donald Ratliff
G. Vijayan *

PDRC 87-03

AD-A182 111

(12)

ON A NODE RANKING PROBLEM OF TREES AND GRAPHS

Ananth. V. Iyer
H. Donald Ratliff
G. Vijayan *

PDRC 87-03

Report by:
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA. 30332

* School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA. 30332

This work was supported in part by the Office of Naval Research
Contract N0014-86-K-0173

On a Node Ranking Problem of Trees and Graphs

Ananth V. Iyer*
H. Donald Ratliff*
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

G. Vijayan
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

October 24, 1986

Abstract

We discuss a problem about ranking nodes of a graph, which is a restriction of the general graph coloring problem. A graph is said to have rank number k if its vertices can be ranked using integers $1, \cdots, k$ such that if two nodes have the same rank i, there is a node with rank greater than i on every path between the two nodes. We give an $O(n \log n)$ algorithm for optimal ranking of trees, and also present bounds on the rank number for trees, planar and other graph classes.

*Authors supported in part by Office of Naval Research Contract N0014-86-K-0173

1 Introduction

Given a graph, we want to assign an integer rank to each node of the graph, such that all paths between two nodes of the same rank pass through at least one node of a higher rank. Imagine the graph being a communication network, then this is equivalent to partitioning the nodes into rank classes, such that any message between nodes of the same rank must pass through a node of a higher rank. Two adjacent nodes cannot receive the same rank, and thus this problem is a restriction of the general coloring problem. The optimal ranking problem is the problem of minimizing the number of distinct ranks. In this paper, we develop an $O(n \log n)$ algorithm for optimal ranking of trees, and also present bounds on the rank number for trees, planar and other graph classes.

Definition 1 A graph G is said to have a *k-ranking* if there exists a ranking of the nodes of G with integers $1, 2, \cdots, k$ satisfying the condition that on every path between two nodes of the same rank there is at least one node with a higher rank. □

Definition 2 The *rank number* $r(G)$ of a graph G is the smallest k such that G has a k-ranking. □

The following lemma can be deduced from the definition of k-ranking.

Lemma 1 There is exactly one node with the largest rank in any ranking of a graph. □

Consider how a connected graph G can be partitioned recursively by removing one node and its incident edges from each of the remaining connected subgraphs until the connected subgraphs have one node in them. Such a recursive decomposition of a graph is called a *1-partition* of the graph, and the tree representing the recursive decomposition is called a *1-partition tree*. Let the height of a rooted tree be the number of edges on the longest path from the root to a leaf in the tree.

Given a 1-partition tree with height k of a graph G, we can rank the nodes of the graph with $k+1$ ranks by simply ranking the node associated with the root of the tree with $k+1$ and recursively ranking each of the connected subgraphs associated with the subtrees rooted at the children of the root with k ranks. It can be easily verified that all paths between two nodes with the same rank must pass through a node with a higher rank. Conversely, given a $(k+1)$-ranking of a graph, one can obtain a 1-partition tree of height k by removing the node with rank $k+1$ first and recursively removing the node of the highest rank from each of the remaining connected subgraphs. Thus we have the following lemma.

Lemma 2 For a graph G, the smallest height of all 1-partition trees $= r(G)-1$. □

A graph G will have a k-ranking if and only if for some vertex v, each of the connected subgraphs of $G-v$ have $(k-1)$-rankings. Thus for fixed k, we can derive an $O(n^{2k})$ polynomial time algorithm for testing if G has a k-ranking.

2 Optimal Ranking of Trees

There is exactly one path between any two nodes in a tree. Trees are bipartite and are 2-colorable, but the only trees that have a 2-ranking are stars (a single node adjacent to a set of independent nodes). However we show in the following that the rank number of trees has a good upper bound.

Lemma 3 Let T be a tree with n nodes. Then $r(T) \leq \log_{3/2} n$.

Proof: It is well known [2] that a tree of n nodes can be partitioned into subtrees each having no more than $2n/3$ nodes by the removal of one node. Recursively applying this operation, we can construct a 1-partition tree of height $h(n)$ given by the recurrence relation $h(n) \leq 1 + h(2n/3)$. From lemma 2, we have $r(T) = h(n) \leq \log_{3/2} n$. □

We now proceed with the development of our algorithm for optimal ranking of trees.

Lemma 4 If a tree T has two node disjoint subtrees T_1 and T_2 each with rank number c, then the rank number of T is at least $c+1$.

Proof: The rank number of T is at least c. Suppose the rank number of T is c. By lemma 1, there is only one node with rank c. Remove this node from T, and now the resulting subtrees all have rank number less than c. Since T_1 and T_2 are node disjoint, one of the resulting subtrees must contain T_1 or T_2, which means that this subtree has rank number c which is a contradiction. □

Definition 3 A ranking of a tree T rooted at a node v is said to be *c-critical* with respect to v if

1. $r(T) = c$
2. The subtree T^c consisting of the node ranked c and its descendants also has rank number c
3. The tree $T - T^c$, obtained by deleting T^c and the edge connecting it to the rest of T, is either empty or is c'-critical with respect to v for some $c' < c$.

The operation giving pruned tree $T - T^c$ from T is called *pruning using the rank* c. □

It follows from part 2 of the definition that, in a c-critical ranking, the rank c cannot be moved up the tree towards the root. From part 3, it follows that the next highest rank in $T - T^c$ also cannot be moved up, and so on. Note that if the root has rank c the tree $T - T^c$ will be empty. Thus if the root is ranked with the rank number then the ranking is automatically critical.

Definition 4 In a critical ranking of a tree T with respect to a root v, the list of decreasing ranks obtained by taking the largest rank and pruning the tree at the node with this rank, and recursively applying this pruning operation on

the remaining tree (if nonempty) is defined as the *list of critical ranks* for the tree T with respect to the root v. □

The notions of critical rankings and lists will allow us to rank a node of a tree without reranking its subtrees. This will become apparent in the proofs of theorems 1 and 2.

Theorem 1 There exists a $r(T)$-critical ranking with respect to any node in a tree T.

Proof: The proof is by induction on the number of nodes n. For $n = 1$ the single node tree trivially has a 1-critical ranking. Assume all trees T' with less than n nodes have an $r(T')$-critical ranking. Let T be a tree with n nodes which is rooted at node v. Let $T_1, \cdots, T_d$ be the subtrees rooted at the children of v. Each T_i has less than n nodes and by inductive hypothesis have $r(T_i)$-critical rankings with respect to their roots. Let T_m have the largest rank number say a among all these subtrees. Let T_m^a be the subtree of T_m rooted at the node w ranked a. By definition of critical rankings $r(T_m^a) = a$.

Consider the tree $T - T_m^a$. Let $r(T - T_m^a) = b$. Note that $b \leq a + 1$ because $r(T) \leq a + 1$. This tree has less than n nodes and by inductive hypothesis has a b-critical ranking with respect to the root v.

If $b \leq a - 1$ then the b-critical ranking of $T - T_m^a$ plus the a-ranking of T^a provides an a-critical ranking of T. If $b = a$ then, by lemma 4, $r(T) > a$, and we can critically rank T with $a + 1$ ranks by taking the critical rankings of the subtrees T_i's and ranking root v with rank $a + 1$. If $b = a + 1$ then the same ranking as above will give an $(a + 1)$-critical ranking of T with respect to v. □

The proof of theorem 1 suggests a recursive algorithm for optimum ranking of a tree. To rank the tree T, the algorithm will recursively rank the subtrees $T_1, \cdots, T_d$. An additional recursion to rank $T - T_m^a$ may be required. Such an algorithm will not satisfy our claimed time complexity mainly because of the need for recursion on the tree $T - T_m^a$. In the following we describe an

algorithm that eliminates the recursive call by using the notion of list of critical ranks introduced earlier.

Consider the following two procedures for critically ranking trees. The *procedure critical_rank_tree* is a recursive procedure that makes calls to *procedure root_rank*. This second procedure takes as input, the lists of critical ranks $L_1, \cdots, L_d$ of the subtrees $T_1, \cdots, T_d$ of the node v, and ranks v appropriately to produce a critical ranking of T with respect to v. It also produces the critical list L of T with respect to v. It is important to note that this procedure does not rerank the subtrees $T_1, \cdots, T_d$. The correctness of this procedure is discussed in theorem 2.

```
Procedure critical_rank_tree (T, v, L);
begin
        if (n = 1) then rank(v) = 1
        else begin
                    Let T_1, ..., T_d be the subtrees of v rooted
                    at its children v_1, ..., v_d;
                    for i := 1 to d do critical_rank_tree (T_i, v_i, L_i);
                    root_rank (v, L_1, ..., L_d, L)
             end
end;
```

```
Procedure root_rank (v, L_1, ..., L_d, L);
begin
        for i := 1 to d do t_i := max L_i;
        a := max{t_1, ..., t_d};
        c := a + 1;
        avail := a + 1;
        L := ∅;
        while v is not ranked do
        begin  c := c − 1;
               if (c = t_i) and (c = t_j) for i ≠ j then
               begin     rank(v) := avail;
                         L := L ∪ {avail}
               end;
               if (c = t_i) for exactly one i then
               begin     L := L ∪ {c};
                         L_i := L_i − {c};
                         t_i := max L_i;
               end;
               if (c ≠ t_i) for any i then avail := c;
        end
end;
```

Theorem 2 Let T be a tree with n nodes and v be any node in the tree. The *procedure critical_rank_tree* produces a $r(T)$-critical ranking of T with respect to v in $O(n \log n)$ time.

Proof: To prove correctness of *procedure critical_rank_tree*, it is enough to show that *procedure root_rank* is correct.

At the beginning of a pass through the *while* loop in the procedure, the current lists L_i are nothing but the critical lists for the subtrees obtained from

the T_i's by successively applying the pruning operations on the T_i's using ranks $a \cdots c+1$. Note for some values in $a \cdots c+1$ pruning may not have occurred in any of the trees. In effect, during each pass through the loop we are pruning the trees with the current value of c. The loop is executed until two of the pruned trees have the same largest rank $c \leq a$. Let the pruned trees at this stage be $T'_1, \cdots, T'_d$. It is easy to see that the tree $T' = v + T'_1 + \cdots + T'_d$ has rank number $c+1$, and we can obtain a $(c+1)$-critical ranking of this tree T' by simply combining the old rankings from the trees T_i projected on the pruned trees T'_i, and giving v rank $c+1$. This ranking is critical for T' because the root v has the highest rank. Suppose, we attach back the last subtree that was pruned off from the collection of trees. If this subtree also has rank number $c+1$, then the new T' has rank number $c+2$ and we can obtain a $(c+2)$-critical ranking by simply reranking v with rank $c+2$. Suppose, we repeat, in reverse order, this operation of attaching back the subtrees that were pruned off, until *either* the next subtree T_{next} to be attached has rank number larger than the current rank on v in the current T', *or*, all subtrees that were pruned off have been attached back.

In the first case, we can obtain a critical ranking of $T' + T_{next}$ without having to rerank v. Now we can attach back the rest of the subtrees that were pruned off without any further reranking of v, and finally end up with a critical ranking of T. In effect, we have given v the smallest rank greater than c which is not in any of the starting lists L_i. In this case $r(T) = a$, and v has received a rank which is less than a. It is easy to verify that *procedure root_rank* assigns exactly the same rank to v using the variable *avail*.

In the second case, T' and the subtrees attached back have the same rank number at each stage. At the final step T' has rank number a and the subtree to be attached also has rank number a, and hence by lemma 4, the rank number of T is $a+1$. In this case, *procedure root_rank* assigns the rank $a+1$ to v and the ranking of T is automatically critical.

We have thus shown that *procedure root_rank* produces a critical ranking of T with respect to v. The list of critical ranks of T with respect to v is simply the list of all ranks in the starting lists L_i which are greater than the rank assigned to v, along with the rank of v. It is easy to verify that list L in the procedure correctly computes this critical list. Thus we have completed the proof of correctness of *procedure root_rank*.

Now we turn our attention to time complexity of the algorithm. Note that the L_i's are in decreasing order and therefore operations of selecting the maximum from a list, and deleting the maximum can be done in constant time. Thus the time complexity of *procedure root_rank* is $O(d \cdot k)$, where d is the degree of v, and k is the rank number of T. The time complexity of *procedure critical_rank_tree* is simply the sum of the time complexities of *procedure root_rank* over all nodes in the tree. The sum of the degrees of nodes in a tree is $2(n-1)$. The rank number of any tree, by lemma 3, is no more than $\log_{3/2} n$. Thus the time complexity of the algorithm is $O(n \log n)$. □

3 Bounds on the Rank Number

In this section we derive upper bounds on the rank number of various classes of graphs. The proofs of the upper bounds are constructive and thus imply algorithms for achieving these upper bounds.

Lemma 3 gives a $\log_{3/2} n$ upper bound on the rank number of any tree. The following two lemmas about rank number of restricted trees can be proved using the equivalence between 1-partitions and rankings.

Lemma 5 The rank number of a path with $2^k - 1$ vertices is k. □

Lemma 6 The rank number of a complete binary tree of height h is $h + 1$. □

We can obtain an upper bound for the rank number of trees in terms of their diameters.

Lemma 7 Let T be a tree with diameter d. Then $r(T) \leq \log_2(d+2)!$.

Proof: From lemma 5, a path of length d can be ranked with at most $\log_2(d+2)$ ranks. Remove from T the path of length d. Each one of the resulting subtrees has diameter at most $d-1$. Rank these subtrees inductively using the same ranks, and then rank the vertices on the removed path with larger ranks. Thus $r(d)$ the rank number of trees of diameter d satisfies the recurrence relation $r(d) \leq r(d-1) + \log_2(d+2)$, and hence $r(d) \leq \log_2(d+2)!$ □

Lemma 8 Let G be a $2^k - 1 \times 2^k - 1$ grid. Then $r(G) \leq 2^{k+2} - 3k - 4$.

Proof: Remove all the $2^{k+1} - 3$ nodes from the center horizontal grid line and the center vertical grid line. Rank the resulting four $2^{k-1} - 1 \times 2^{k-1} - 1$ subgrids recursively using the same set of ranks. Rank the removed nodes with larger distinct ranks. Since any path between two nodes from different subgrids must pass through one of the reemoved nodes, the ranking so obtained is a feasible ranking. Thus, we have $r(k) \leq r(k-1) + 2^{k+1} - 3$. It is easy to show that $r(k) \leq 2^{k+2} - 3k - 4$. □

Hence, for a $\sqrt{n} \times \sqrt{n}$ grid, the rank number is $O(\sqrt{n})$.

Lemma 9 The rank number of the complete graph K_n is n. □

Lemma 10 The rank number of the complete bipartite graph $K_{m,n}$ with $n \leq m$ is $n+1$.

Proof: Suppose two nodes u, v in the m-part receive the same rank i. All nodes in the n-part are on paths of length two between u and v, and hence must receive ranks at least $i+1$. If two nodes x, y in the n-part receive the same rank $j > i$ then all ranks in the m-part including the rank i must be larger than j, which is a contradiction. Hence all ranks in one of the parts must be identical. We can rank all vertices in the m-part with distinct ranks $2, \cdots, m+1$, and rank all vertices in the n-part with rank 1. □

We can obtain upper bounds on the rank number of classes of graphs using their separator property [3].

Definition 5 A class of graphs, closed under subgraph property, is said to be *$f(n)$-separable*, if for any graph G with n vertices in the class, there exists a set of less than $f(n)$ vertices whose removal results in subgraphs G_1 and G_2, each having no more than $2n/3$ vertices. □

In the proof of lemma 3, we used the fact that forest of trees are 1-separable. Planar graphs are $O(\sqrt{n})$-separable [3]. Outerplanar graphs are known to be $O(1)$-separable [1].

Lemma 11 Let G be a member of a class of graphs which are $f(n)$-separable. The rank number $r(n)$ of G satisfies the recurrence $r(n) \leq f(n) + r(2n/3)$.

Proof: Use a technique similar to the one in the proofs of lemmas 8. □

Lemma 12 The rank number of any planar graph with n nodes is $O(\sqrt{n})$.

Proof: Planar graphs are $c\sqrt{n}$-separable, for some constant c. By lemma 11, we have $r(n) \leq c\sqrt{n} + r(2n/3)$. It can be easily shown that $r(n)$ is $O(\sqrt{n})$. □

Lemma 13 The rank number of any outerplanar graph with n nodes is $O(\log n)$.

Proof: Outerplanar graphs are $O(1)$-separable, hence by lemma 11, their rank number is given by $r(n) \leq O(1) + r(2n/3)$. Hence $r(n)$ is $O(\log n)$. □

References

[1] C. E. Leiserson, "Area-efficient graph layouts (for VLSI)," *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*, October 1980, pp. 270-281.

[2] P. M. Lewis, R.E. Stearns, and J. Hartmanis, "Memory bounds for recognition of context-free and context-sensitive languages," *Proceedings of the 6th IEEE Annual Symposium on Switching Theory and Logic Design*," 1965, pp. 191-202.

[3] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal of Applied Mathematics*, Vol. 36, No. 2, 1979, pp. 177-189.

END

8-87

DTIC