Optimal *c*-Vertex-Ranking of Weighted Trees

Md. Abul Kashem*, Mohammad Masud Hasan*, and Sheikh Mohammed Nazrul Alam*

Abstract: A *c-vertex-ranking* of a graph *G*, for a positive integer *c*, is a labeling of the vertices of *G* with integers such that, for any label *i*, deletion of all vertices with labels > *i* leaves connected components, each having at most *c* vertices with label *i*. In this paper we newly define a *c*-vertex-ranking of a weighted graph *G* as follows: let (*G*, *w*) be a weighted graph, where *w* is an assignment of positive weights to the vertices of *G;* we label the vertices of (*G*, *w*) in such a way that it is a *c*-vertex-ranking of *G* and the sum of the maximum weights of vertices in each rank classes is minimized. To be precise, we want to minimize $w_\varphi = \sum_{i=1}^{\alpha} w_i$, where w_i = max{$w(u)$: $u \in V(G)$ is labeled with ranki $\varphi(u) = i$ under a *c*-vertex-ranking φ}, and α is the number of ranks used by φ. A *c*-vertex-ranking φ is optimal if w_φ is as small as possible. We present an algorithm to find an optimal *c*-vertex-ranking of a given weighted tree (*T*, *w*) in time $O(\alpha\, c^{2\alpha}\, n^{\alpha+1})$, where *n* is the number of the vertices in *T*.

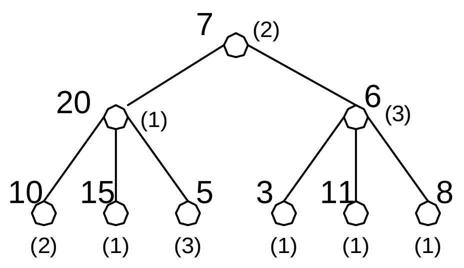
Keywords: Algorithm, *c*-Vertex-ranking, Tree, Visible vertices, Weighted graph.

1 Introduction

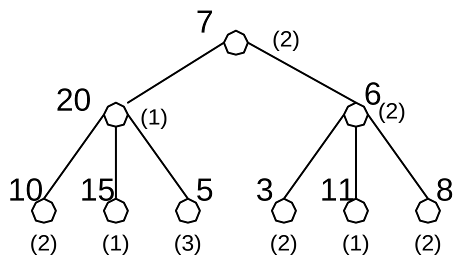
For a positive integer *c*, a *c-vertex-ranking* of a graph *G* is a labeling of the vertices with integers such that, for any label *i*, deletion of all vertices with labels > *i* leaves connected components, each having at most *c* vertices with label *i* [12]. The *c*-vertex-ranking problem is to find a *c*-vertex-ranking of a given graph using minimum number of ranks. The problem is NP-hard in general [12], while Zhou *et al.* have obtained a linear-time algorithm to solve the *c*-vertex-ranking problem for trees [12]. Then Kashem *et al.* have presented a polynomial-time algorithm to solve the *c*-vertex-ranking problem for partial *k*-trees, that is graphs of treewidth bounded by a fixed integer *k* [6]. Recently, Kashem *et al.* have obtained an $O(n^9 \log^9 n \log\log n)$ time algorithm for solving the *c*-vertex-ranking problem on series-parallel graphs [5].

An *ordinary vertex-ranking* of a graph *G* is a labeling (ranking) of vertices of *G* with integers such that any path between two vertices with the same label *i* contains a vertex with label *j* > *i* [4]. Clearly an ordinary vertex-ranking is a 1-vertex-ranking. Although the ordinary vertex-ranking problem is NP-hard in general [2, 9], Iyer *et al.* presented an $O(n \log n)$ time algorithm to solve the ordinary vertex-ranking problem for trees[4] , where *n* is the number of vertices of the input tree. Then Schaffer obtained a linear-time algorithm by refining their algorithm and its analysis [10]. Deogun *et al.* gave algorithms to solve the ordinary vertex-ranking problem for interval graphs in $O(n^3)$ time and for permutation graphs in $O(n^6)$ time [3]. Recently, Newton *et al.* presented an $O(n^3)$ time algorithm to solve the ordinary vertex-ranking problem for permutation graphs [8]. Kloks *et al.* have obtained an algorithm for computing the vertex-ranking number of an asteroidal triple-free graph in time polynomial in the number of vertices and the number of minimal separators [7].

In this paper, we newly define a generalization of the *c*-vertex-ranking on weighted graphs. Let *G* = (*V*, *E*) be a graph, and let *w*: $V \rightarrow \mathbf{N}$ be an assignment of positive weights to the vertices of *G*. From here on we denote by (*G*, *w*) a graph with weight *w*. We consider the following *c*-vertex-ranking problem of the weighted graph (*G*, *w*): for a *c*-vertex-ranking φ of *G*, we denote by $w_\varphi(i)$ the maximum weight of the vertices having rank *i* and denote by w_φ the sum $\sum_{i=1}^{\alpha} w_\varphi(i)$ of these maximum weights, where α is the number of ranks used by φ. Let $\chi(G, w, \alpha)$ be the minimum of w_φ among all *c*-vertex-rankings φ using α ranks, and let $\chi(G, w)$ be the minimum among $\chi(G, w, \alpha)$ for all $\alpha \geq 1$. We say



(a)one optimal 2-vertex-ranking of (T,*w*)



(c) another optimal 2-vertex-ranking of (*T*,*w*)

Figure 1: Two optimal 2-vertex-rankings of a weighted tree (*T*,*w*)

*Department of Computer Science and Engineering,Bangladesh University of Engineering and Technology, Dhaka 1000. Email: kashem@cse.buet.edu the c-vertex-ranking φ of G is optimal if $w_\varphi = \chi(G, w)$. The optimal c-vertex-ranking can be found using ranks $\le \alpha$ for some α, $1 \le \alpha \le n$. It is possible that two optimal c-vertex-rankings of a weighted graph (G, w) use different number of ranks. Figure 1 depicts two optimal 2-vertex-rankings of a weighted tree (T,w) using three ranks, where ranks are drawn in parentheses and weights next to the vertices.

The problem of finding an optimal c-vertex-ranking of a graph has applications in scheduling the parallel assembly of a complex multi-part product from its components, where the vertices of the graph correspond to the parts and edges correspond to assembly operations [2, 11, 12]. In practical fields, the required time for assembling different parts of a product is not same, that is, the weight of each vertex is not same. So the c-vertex-ranking problem on unweighted graphs is not suitable for practical applications. In that case we have to consider a weighted graph, where weight of each vertex represents required time to complete the task associated with the corresponding vertex. Our proposed problem is the theoretical interpretation of this type of practical problems.

2 Preliminaries

In this section we define some terms and present easy observations. Let $T = (V, E)$ be a tree with vertex set V and edge set E. We often denote by $V(T)$ and $E(T)$ the ssvertex set and the edge set of T, respectively. We denote by n the number of vertices in T. T is a 'free tree', but we regard T as a 'rooted tree' for convenience sake: an arbitrary vertex of tree T is designated as the root of T. We use the notations as: root, internal vertex, child and leaf in their usual meaning. An edge joining vertex u and v is denoted by (u, v). The degree of a vertex $u \in V$ is denoted by $d(u)$. The maximal subtree of sT rooted at a vertex $v \in V$ is denoted by $T(v)$. For a c-vertex-ranking φ of tree T and a subtree T' of T, we denote by $\varphi|T'$ a restriction of φ to $V(T')$: let $\varphi' = \varphi|T'$, then $\varphi'(v) = \varphi(v)$ for all $v \in V(T')$.

The number of ranks used by a c-vertex-ranking φ of tree T is denoted by $\#\varphi$. One may assume without loss of generality that φ uses the consecutive integers 1, 2, 3, …, $\#\varphi$ as the ranks. A vertex v of T and its rank $\varphi(v)$ are visible (from the root under φ) if all the vertices in the path from root to v have ranks $\le \varphi(v)$. Thus the root of T and $\#\varphi$ are always visible. We denote by $L(\varphi)$ the list of ranks of all visible vertices, and call $L(\varphi)$ the list of c-vertex-ranking φ of the rooted tree T. For an integer γ we denote by $count(L(\varphi), \gamma)$ the number of γ's contained in $L(\varphi)$, i.e. the number of visible vertices of rank γ. One can easily observe that $count(L(\varphi), \gamma) \le c$ for each rank γ. We then have the following lemma [6,12].

Lemma 2.1 *Let T be a tree, and let u be a vertex in T. Then a vertex-labeling φ of $T(u)$ is a c-vertex-ranking of $T(u)$ if and only if*

(a) at most c vertices of the same rank are visible from u under φ in T, that is, $count(L(\varphi),\gamma) \le c$ for each $\gamma \in L(\varphi)$; and

(b) if u is an internal node in T and has d children v_1, v_2, ... ,v_d, then $\varphi|T(v_i)$ is a c-vertex-ranking of $T(v_i)$ for each i, $1 \le i \le d$. □

For a list L and an integer i, we define a sublist $[i \le L]$ of L as follows:

$$[i \le L] = \{l \in L \mid i \le l\}.$$

Similarly we define sublists $[i < L]$, $[L < i]$ and $[L \le i]$ of L. For lists L_1 and L_2 we use $L_1 \subseteq L_2$ and $L_1 \cup L_2$ in their usual meanings in which we regard L_1, L_2 and $L_1 \cup L_2$ as multi-sets.

We transform the tree T to a *binary decomposition tree* T_b as follows: Regard T as a rooted tree by choosing an arbitrary node as the root, and replace every internal node u having d children, say v_1, v_2, … v_d with $d + 1$ new

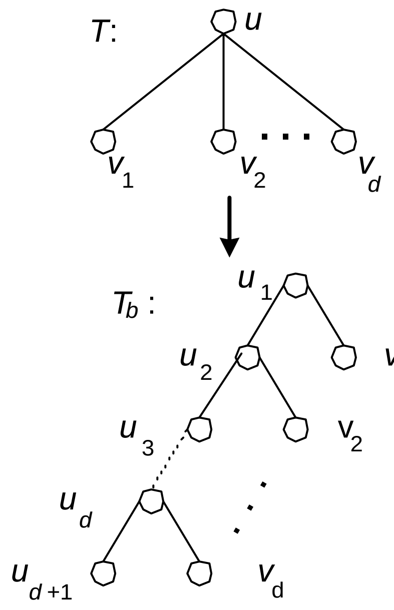


Figure 2: Illustration of the binary transformation

nodes $u_1, u_2, \ldots, u_{d+1}$, such that $w(u_{d+1}) = w(u)$, where u_i, $1 \le i \le d$, is the father of u_{i+1} and the i-th child u_i of u, and u_{d+1} is a leaf of tree T_b [1, 6]. (See Fig 1). This transformation can be done in $O(n)$ time [1, 6]. The resulting binary decomposition tree T_b has the following characteristics:

- the number of nodes in T_b is $O(n)$;
- for each internal node u having d children in T, there is exactly one leaf u_{d+1} in T_b such that $w(u_{d+1}) = w(u)$.
- each leaf x in T_b corresponds to a node u in T such that $w(x) = w(u)$.
- each internal node x in T_b is a dummy node with no weight.

We associate a subgraph $T_x = (V_x, E_x)$ of T with each node x of T_b, where

$V_x = \{ y | y \text{ is a leaf in } T_b(x) \}$, and

$E_x = \{ (u,v) \in | u,v \in V_x \}$

Thus T is associated with the root of T_b.

3 Optimal *c*-vertex-ranking of a weighted tree

The main result of this paper is the following theorem.

Theorem 3.1 *Let R be a set of ranks such that a weighted tree* (T, w) *has an optimal c-vertex-ranking using* $\alpha = |R|$ *ranks. Then an optimal c-vertex-ranking of a weighted tree having n vertices can be found in time* $O(\alpha c^{2\alpha} n^{\alpha+1})$. □

If we assume that α is a constant, we then have the following corollary.

Corollary 3.2 *An optimal c-vertex-ranking of a weighted tree can be found in polynomial-time if* α *is constant.* □

Let α be the minimum number of ranks needed for an optimal *c*-vertex-ranking of a weighted tree (T, w). In the remaining of this section we give an algorithm to find an optimal *c*-vertex-ranking of a weighted tree (T, w) in time $O(\alpha c^{2\alpha} n^{\alpha+1})$. Our algorithm uses the technique of 'bottom-up-tree computation'.

We say a sequence $(a_1, a_2, \dots, a_\alpha)$ of integers is a *valid sequence* provided each a_i is the weight of a vertex of tree T, or $a_i = 0$ ($a_i = 0$ means that no vertex has rank i). Since there are at most $n + 1$ different choices for the value of a_i, the number of valid sequences is at most $(n + 1)^\alpha$. We first assume that $(a_1, a_2, \dots, a_\alpha)$ is a fixed valid sequence. Our algorithm checks whether there exists a *c*-vertex-ranking φ of (T, w) such that $w_\varphi(i) \le a_i$. We call such *c*-vertex-rankings of (T, w) as *good c-vertex-rankings* with respect to the valid sequence $(a_1, a_2, \dots, a_\alpha)$. Thus the algorithm checks all valid sequences and finds a sequence for which $w_\varphi = \sum_{i=1}^{\alpha} a_i$ is minimum and has at least one *c*-vertex-ranking. A good *c*-vertex-ranking φ for which w_φ is minimum is the desired optimal *c*-vertex-ranking.

Let T_b be the binary decomposition tree of T. We then have the following two lemmas.

Lemma 3.3 *Let x be a leaf in* T_b *that corresponds to a vertex u in T, and let* φ *be a good c-vertex-ranking of* (T, w) *with respect to a valid sequence* $(a_1, a_2, \dots, a_\alpha)$. *Then there exists a rank j,* $1 \le j \le \alpha$, *such that* $\varphi(u) = j$ *and* $w(u) \le a_j$.

Proof: For a contradiction, let u has rank $\varphi(u) = j$ under the *c*-vertex-ranking φ such that $w(u) > a_j$. Then

$$w_\varphi = a_1 + a_2 + \dots + a_{j-1} + w(u) + a_{j+1} + \dots + a_\alpha$$
$$> a_1 + a_2 + \dots + a_{j-1} + a_j + a_{j+1} + \dots + a_\alpha .$$

So the *c*-vertex-ranking φ gives larger weight summation than $\sum_{i=1}^{\alpha} a_i$, which is not acceptable. □

Lemma 3.4 *Let x be an internal node in* T_b *with two children y* and *z. Let* η *and* ψ *be the c-vertex-rankings of* T_y *and* T_z, *respectively. Let* φ *be the vertex-labeling of* T_x *extended from* η *and* ψ. *Let x and y be the dummy vertices in* T_b *that corresponds to a vertex u in T, and let z be the dummy vertex in* T_b *corresponds to a vertex w in T. Then* $L(\varphi, u) = L(\eta, u) \cup [\eta(u) \le L(\psi, w)]$.

Proof: Here $\eta = \varphi|T_y$ and $\psi = \varphi|T_z$ and $\varphi(u) = \eta(u)$. Then clearly

$$L(\varphi, u) = L(\eta, u) \cup [\eta(u) \le L(\psi, w)]. \quad □$$

We then have the following algorithm for *c*-vertex-ranking of a weighted tree (T, w).

```
Procedure c_vertex ( )
  begin
1.  decompose the tree $T$ to a binary decomposition
    tree $T_b$, and let $r$ be the root of $T_b$;
2.  for each valid sequence $(a_1, a_2, \dots, a_\alpha)$ do
    begin
3.    ranking($T_b(r)$, $a_1, \dots, a_\alpha$);
4.    check whether there exists a c-vertex-ranking
      at the root;
    end
5.  find the optimal sequence for which $\sum_{i=1}^{\alpha} a_i$ is
    minimum and has a c-vertex-ranking;
6.  find a c-vertex-ranking of $T$ having the optimal
    sequence;
  end


Procedure ranking($T_b(x)$, $a_1, a_2, \dots, a_\alpha$)
  begin
7.  if $x$ is a leaf then
    begin
      let $u$ be a vertex in $T$ that corresponds to $x$ in $T_b$;
8.    for each $a_i \ge w(u)$ do
        begin
9.        a trivial ranking $\varphi(u) \leftarrow i$;
10.       $L(\varphi, u) = \{i\}$;
        end
    end
    else
    begin
11.   let $y$ and $z$ be the two children of $x$; let $x$ and $y$ be
      the dummy vertices in $T_b$ that corresponds to a
      vertex $u$ in $T$, and let $z$ be the dummy vertex in
      $T_b$ corresponds to a vertex $w$ in $T$.
12.   ranking($T_b(y)$, $a_1, a_2, \dots, a_\alpha$);
13.   ranking($T_b(z)$, $a_1, a_2, \dots, a_\alpha$);
14.  for each visibility list $L(\eta, u)$ do
15.     for each visibility list $L(\psi, w)$ do
        begin
16.       find a visibility list $L(\varphi, u)$ from $L(\eta, u)$
          and $L(\psi, w)$ using Lemma 3.4;
```

```
17.            check whether $\varphi$ is a $c$-vertex-ranking by
             Lemma 2.1;
          end
      end
18.  if $\varphi$ is a $c$-vertex-ranking then
19.     return $L(\varphi, u)$;
   end.
```

Lemma 3.5 *Let R be a set of ranks such that a weighted tree* (T, w) *has an optimal c-vertex-ranking using* $\alpha = |R|$ *ranks. Then the number of possible distinct visibility lists at any node u in* T_b *is at most* $(c + 1)^{\alpha}$.

Proof: Since $|R| = \alpha$ and $0 \le count(L(\varphi),\ j) \le c$ for a c-vertex-ranking φ and a rank $j \in R$, clearly the number of distinct visibility lists $L(\varphi)$ at any node u in T_b is at most $(c + 1)^{\alpha}$. □

Line 1 runs in linear-time [1, 6]. Line 2 is executed $O((n + 1)^{\alpha}) = O(n^{\alpha})$ times. Each execution of Line 2 calls Line 3 for procedure *ranking* once. Line 8 runs $O(\alpha)$ times for each leaf. Clearly Lines 9 and 10 can be done in constant time. Thus Lines 7-10 can be done in time $O(\alpha n)$ for all leaves. Lines 12 and 13 are executed $O(n)$ times taking recursion into account. Line 14 executes $O((c + 1)^{\alpha})$ times and for each execution of Line 14, Line 15 executes $O((c + 1)^{\alpha})$ times, so Lines 16 and 17 are executed $O((c + 1)^{2\alpha})$ times. Each execution of Lines 16 and 17 take $O(\alpha)$ time. Clearly Lines 18 and 19 take constant time.
Thus procedure *ranking* takes $O((c + 1)^{2\alpha}.n\alpha) = O(\alpha c^{2\alpha}n)$ time in total. Each execution of Line 4 takes constant time. Therefore Lines 2-4 need $O(\alpha n c^{2\alpha}.n^{\alpha}) = O(\alpha c^{2\alpha} n^{\alpha+1})$ time in total. Lines 5 and 6 can be done in time $O(\alpha c^{2\alpha} n^{\alpha+1})$. Thus the time complexity of our algorithm is $O(\alpha c^{2\alpha} n^{\alpha+1})$.

4 Conclusions

We newly define a c-vertex-ranking of a weighted graph, and give an algorithm to find an optimal c-vertex-ranking of a given weighted tree (T, w) in time $O(\alpha c^{2\alpha} n^{\alpha+1})$ for any positive integer c, where n is the number of vertices in T and α is the number of ranks used.

If α is constant then our algorithm runs in polynomial-time. But when α is not constant then the algorithm does not have polynomial characteristics. In fact due to the lack of any local optimality it is not possible to use dynamic-programming algorithm in this problem. So, there is a chance of the problem being NP-complete for trees. We close the paper with the following open questions.

1. Can α has constant or logarithmic upper bound ?
2. Is the c-vertex-ranking problem NP-complete for weighted trees ?

References

[1] H. L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, J. Algorithms, 11 (1990), pp. 631-633.

[2] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Mullar, Zs Tuza, Rankings of graphs, SIAM Journal on Discrete Math. 21 (1998), pp. 168-181.

[3] J. S. Deogun, T. Kloks, D. Kratsch, H. Mullar, On vertex ranking for permutation and other graphs, Proc. 11th Ann. Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Vol. 775, Springer-Verlag, 1994, pp.747-758.

[4] A. V. Iyer, H. D. Ratliff, G.Vijayan, Optimal nodes ranking of trees, Information Processing Letters 28 (1988) 225-229.

[5] M. A. Kashem , M. Ziaur Rahman , An efficient algorithm for optimal c-vertex-ranking of series-parallel graphs, Proc. of the 2nd. Int. Conf. on Computer and Information Technology(ICCIT′99)(1999), pp.309-314

[6] M. A. Kashem, X. Zhou, T. Nishizeki, Algorithms for generalized vertex-ranking of partial k-trees, Theoretical Computer Science 240 (2000), pp. 407-427.

[7] T.Kloks, H.Muller, C.K. Wong , Vertex ranking of asteroidal triple-free graphs, Proc. of the 7th Int. Symp. on Algorithms and Computation (ISSAAC′96), Lecture Notes in Computer Science, 1178(1996), pp.178-182.

[8] A. H. Newton, M. A. Kashem, An efficient algorithm for optimal vertex-ranking of permutation graphs, Proc. 2nd. Int. Conf. on Computer and Information Technology(ICCIT99) (1999), pp. 315-320.

[9] A. Pothen, The Complexity of optimal elimination trees, Technical Report CS-88-13, Pennsylvania State University, USA, 1988.

[10] A. A. Schaffer, Optimal node ranking of trees in linear time, Information Processing Letters 33 (1989/90) pp. 91-96.

[11] X. Zhou, M. A. Kashem, T. Nishizeki, Generalized edge-ranking of trees, IEICE Trans. on Fundamentals, E81-A.No.2 (1998), pp .

[12] X. Zhou, N. Nagai, T. Nishizeki, Generalized vertex-ranking of trees, Information Processing Letters 56 (1995), pp. 321-328.