An Efficient On-Line Algorithm for Edge-Ranking of Trees

MUNTASIR RAIHAN RAHMAN[1]
MD. ABUL KASHEM[2]
MD. EHTESAMUL HAQUE[2]

[1]David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
[2]Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka 1000, Bangladesh
[1]mr2rahman@cs.uwaterloo.ca
[2]{kashem,ehtesam}@cse.buet.ac.bd

Abstract. An edge-ranking of a graph G is a labeling of the edges of G with positive integers such that every path between two edges with the same label γ contains an edge with label $\lambda > \gamma$. In the on-line edge-ranking model the edges $e_1, e_2 \ldots, e_m$ arrive one at a time in any order, where m is the number of edges in the graph. Only the partial information in the induced subgraph $G[\{e_1, e_2, \ldots, e_i\}]$ is available when the algorithm must choose a rank for e_i. In this paper, we present an on-line algorithm for ranking the edges of a tree in time $O(n^2)$, where n is the number of vertices in the tree.

Keywords: Algorithm, Edge-ranking, Graph, Tree, Visible Edge.

(Received November 27, 2007 / Accepted May 26, 2008)

1 Introduction

An *edge-ranking* of a graph $G = (V, E)$ is an edge-labeling $\varphi : E \to \mathbb{N}$ such that every path in G between two edges with the same label γ contains an internal edge with label $\geq \gamma + 1$. The integer label $\varphi(e)$ of an edge e is called the *rank* of the edge. Clearly an edge-labeling is an edge-ranking if and only if, for any label γ, deletion of all edges with labels $> \gamma$ leaves connected components, each having at most one edge with label γ. Figure 1 shows an edge-ranking of a tree T using 5 ranks.

An edge-ranking of G using the minimum number of ranks(labels) is called an *optimal edge-ranking* of G. The *edge-ranking problem* is to find an optimal edge-ranking of a given graph G. The optimal edge-ranking problem has important applications in scheduling the assembly steps in manufacturing a complex multi-part product [3]. Since the constraints for the edge-ranking problem imply that two adjacent edges cannot have the same rank, the edge-ranking problem is a restriction of the edge-coloring problem.

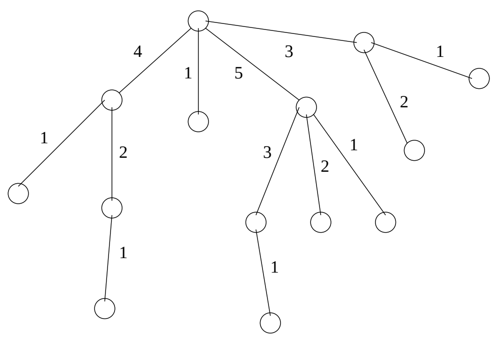

Figure 1: An edge-ranking of a tree T.

The edge-ranking problem is $\mathcal{NP}$-Complete in general [9], although polynomial-time algorithms have been found for trees. Iyer *et al.* [3] gave an $O(n \log_2 n)$ time sequential approximation algorithm for finding an edge-ranking of a tree T using at most twice the minimum number of ranks, where n is the number of vertices in T. Later Torre *et al.* [14] gave an exact algorithm to solve the edge-ranking problem on trees in time $O(n^3 \log_2 n)$ by means of a two-layered greedy method. Recently Lam *et al.* have given a linear-time algorithm for solving the edge-ranking problem on trees [10]. In [14] Torre *et al.* have given a parallel algorithm for solving the edge-ranking problem on trees in $O(\Delta^4 \log^3 n)$ parallel time using $O(2^{\Delta} n^{\Delta+1})$ operations on the CREW PRAM model.

Generalization of the edge-ranking problem was introduced in [15]. For a positive integer c, a *c-edge-ranking* of a graph G is a labeling of the edges of G with positive integers such that, for any label γ, deletion of all edges with labels $> \gamma$ leaves (connected) components, each having at most c edges with label γ [15]. Clearly an ordinary edge-ranking is a 1-edge-ranking. The *c-edge-ranking problem* is to find an optimal c-edge-ranking of a given graph G. Zhou *et al.* gave an algorithm to find an optimal c-edge-ranking of a given tree T for any positive integer c in time $O(n^2 \log \Delta)$, where Δ is the maximum vertex-degree of T [15]. A polynomial-time sequential algorithm and an $O(\log n)$ time parallel algorithm for solving the c-edge-ranking problem on partial k-trees with small treewidth for any positive integer c was given by Kashem *et al.* [5].

The vertex-ranking problem [2] and the c-vertex-ranking problem [16] for a graph G are defined similarly. Iyer *et al.* presented an $O(n \log n)$ time algorithm to solve the vertex-ranking problem on trees [2]. Then Schäffer obtained a linear-time algorithm by refining their algorithm and its analysis [12]. On the other hand, Zhou *et al.* have obtained an $O(c \cdot n)$ time sequential algorithm to solve the c-vertex-ranking problem for trees [16]. Kashem *et al.* gave a polynomial-time sequential algorithm and $O(\log n)$ time parallel algorithm for solving the c-vertex-ranking problem on partial k-trees [6]. Recently, Kashem *et al.* gave an $O(\log n)$ time optimal parallel algorithm for solving the c-vertex-ranking problem on trees [4].

An algorithm is called *off-line* if all input data must be accessed before the output is produced. Most research done in graph theory concentrates on off-line algorithms. The edge-ranking and vertex-ranking algorithms mentioned above are all off-line algorithms. On the contrary, an *on-line* algorithm has to make (partial) decisions after seeing only a subset of the input. For example, for ranking (coloring) problems, there are two natural on-line models: the input is given either vertex-by-vertex or edge-by-edge. The algorithm assigns a rank (color) to the current vertex or edge based only on past history and a rank (color) assigned to a vertex or edge cannot be changed later.

In this paper we are concerned with on-line rankings of graphs. Schiermeyer *et al.* characterized the class of graphs for which on-line vertex-ranking can be found using only three ranks [13]. They also proved that for $n \geq 2$, the greedy first-fit coloring heuristic can rank an n-vertex path using a maximum of $3 \log_2 n$ ranks, independently from the arriving order of vertices. Bruoth *et al.* [1] improved this result by showing that the maximum number of ranks required for on-line ranking the vertices of an n-vertex path is $2\lfloor \log_2 n \rfloor + 1$, where $n \geq 2$. They also obtained a similar bound for cycles. However none of these two papers provide efficient on-line algorithms for vertex-ranking of graphs. Recently Lee *et al.* gave the first on-line vertex-ranking algorithm for trees that runs in $O(n^3)$ time [7]. They also presented an on-line parallel algorithm for ranking the vertices of a tree in time $O(n \log^2 n)$ using $O(n^3 / \log^2 n)$ processors on the CREW PRAM model [8].

In this paper we provide an $O(n^2)$ time on-line algorithm for ranking the edges of a tree, where n is the number of vertices in the tree. In an on-line setting, since the edges arrive one at a time in each iteration, only partial or incomplete information about the input graph is available at each step. So it is not possible to guarantee that an on-line algorithm can rank the edges of a graph with the minimum number of ranks. Therefore we use a greedy strategy in our algorithm to rank the newly arrived edge in each iteration with the least possible rank.

2 Preliminaries

Let $T = (V, E)$ be a tree. We denote $V(T)$ and $E(T)$ as the set of vertices and the set of edges in T, respectively. Let $|V(T)| = n$ and $|E(T)| = m$. When u and v are the endpoints of an edge $e = (u, v)$, they are adjacent and are *neighbors*. In that case, e is said to be incident to u and v. Two edges are adjacent if they have a common endpoint. We denote the degree of any vertex $v \in V(T)$ by $d(v)$. Also for any two edges $e, e' \in E(T)$, we denote the unique path from e to e' by $P(e, e')$. The unique path from a vertex v to an edge e is denoted by $P(v, e)$.

Let e_i be the newly arrived edge at the ith iteration of an on-line algorithm. We denote T_i as the subgraph of T induced by $\{e_1, e_2, \ldots, e_i\}$. Let $T(e_i)$ be the (connected) component of T_i which contains e_i, the newly arrived edge. Only $T(e_i)$ will be considered while ranking e_i, since the edges in other components will not affect the ranks of edges in $T(e_i)$.

Let φ be an edge-labeling of a graph G with positive integers. We denote the rank(label) of an edge e by $\varphi(e)$. The concepts of visible rank and visibility list were introduced by Iyer et al. [2]. Consider any edge $e \in E(T(e_i)) \setminus \{e_i\}$. The rank $\varphi(e)$ of e is said to be *visible* from a vertex $v \in V(T(e_i))$ under φ, if all the edges on $P(v, e)$ are labeled and have ranks $\leq \varphi(e)$. Such an edge e is then called a *visible edge*. The list of all ranks visible from a vertex v under φ is called the *visibility list of* v, and is denoted by $L(v)$. Let $e_i = (u, v)$, and let $L(e_i) = L(u) \cup L(v)$. Then we say that $L(e_i)$ is the *visibility list of* e_i. The list $L(e_i)$ will generally be a multi-set, where an element γ in $L(e_i)$ can appear more than once. A rank that is not visible from an endpoint of e_i under φ is called an *invisible rank*. For any integer γ we denote by $count(L(e_i), \gamma)$ the number of γ's contained in $L(e_i)$ [16].

3 On-Line Edge-Ranking of Trees

The following theorem is the main result of this paper.

Theorem 1 *The edges of a tree T can be ranked using an on-line algorithm in $O(n^2)$ time, where n is the number of vertices in T.*

In the remainder of this section we prove Theorem 1 by giving an on-line algorithm for ranking the edges of a tree T in time $O(n^2)$. It is based on the greedy first-fit coloring heuristic. At the ith iteration, the algorithm takes as input the newly arrived edge e_i. We then rank e_i with the least possible rank without violating the edge-ranking property. To rank e_i, we construct the visibility list $L(e_i)$ by searching in $T(e_i)$ to find all visible ranks. The search is carried out by a recursive depth first search traversal. During the search, we keep track of the largest rank of an edge on a path starting from endpoints of e_i seen so far. As the traversal continues along a path, if an edge is traversed that has a rank greater than the current maximum, then that rank is added to $L(e_i)$ and the largest rank is updated. Since any edge adjacent to e_i is trivially visible from an endpoint of e_i under φ, its rank must belong to $L(e_i)$. To incorporate this case into the algorithm, the largest rank is set to 0 at the beginning of the search. As a result when an edge e adjacent e_i is traversed, its rank $\varphi(e)$ will be trivially greater than 0 and added to $L(e_i)$. The pseudo-code of the algorithm is given below.

Algorithm *On_line_Edge_Ranking_Tree*
begin
  for $i = 1$ to m **do** $\{m = |E(T)|\}$
    read a new edge e_i;
    let $E' = \{e'_1, e'_2, \ldots, e'_p\}$ be the set of edges adjacent to e_i in $\{e_1, e_2, \ldots, e_{i-1}\}$;
    $L := \emptyset$; {Currently L is the visibility list of e_i, that is $L = L(e_i)$}
    RankEdge(e_i, E');
end

Procedure RankEdge(e, E')
begin
  for $j = 1$ to p **do** $\{p = |E'|\}$
    BVL(e, e'_j, 0);
  find minimum α such that $\alpha \notin L$ and $count(L, \beta) \leq 1$, for each β satisfying $\alpha + 1 \leq \beta \leq max\{L\}$;
  $\varphi(e) := \alpha$; {rank e with α}
end

Procedure BVL(e, e', r_{max})
begin
  if $\varphi(e') > r_{max}$ **then**
    $L := L \cup \{\varphi(e')\}$;
  let $E(e')$ be the set of edges adjacent to e' in $T(e_i)$;
  if $(E(e') \setminus \{e\}) \neq \emptyset$ **then**
    for each edge $e'' \in (E(e') \setminus \{e\})$ **do**
      BVL(e', e'', $max\{r_{max}, \varphi(e')\}$);
end

We now prove the correctness of the algorithm.

When $i = 1$, that is, when the first edge e_1 arrives, it does not have any adjacent edges, and so it can be trivially ranked with 1 without violating the edge-ranking property. When $i > 1$, we inductively assume that the edges $e_1, \ldots, e_{i-1}$ have been properly ranked in the previous $i - 1$ iterations. We now prove that e_i is properly ranked at the ith iteration. At first we show that the visibility list is correctly constructed at the ith iteration of the algorithm. We have the following lemma.

Lemma 1 *Let $e_i \in E(T)$ be the newly arrived edge at the ith iteration, and let L be the list constructed by On_line_Edge_Ranking_Tree for the edge e_i. Then $L = L(e_i)$, that is,*

(*i*) *L contains all the ranks visible from an endpoint of e_i under φ; and*

(*ii*) *L does not contain any rank invisible from both endpoints of e_i under φ.*

Proof. (*i*) For a contradiction, assume that γ is a rank visible from an endpoint v of e_i under φ but $\gamma \notin L$. Let e be a visible edge with rank $\varphi(e) = \gamma$, where $e \in E(T(e_i)) \setminus \{e_i\}$. Since γ is visible from the endpoint v

of e_i, all the edges on $P(v, e)$ have ranks $\leq \gamma$. Let e' be the edge incident to v on $P(v, e)$ and e'' be the edge adjacent to e on $P(v, e)$. Since φ is a vertex-ranking, we have $\varphi(e''') < \gamma$ for all edges $e''' \in E(P(e', e''))$. Thus the largest rank seen so far from e' to e'' on $P(v, e)$ is $< \gamma$. So when e will be traversed on $P(v, e)$, we have $\varphi(e) = \gamma > r_{max}$. So γ must be added to L. This contradicts $\gamma \notin L$. Thus L contains all the ranks visible from an endpoint of e_i.

(*ii*) For a contradiction, assume that L contains a rank γ that is invisible from both endpoints u and v of e_i under φ. Let e be a vertex with $\varphi(e) = \gamma$. Let e' be the edge incident to v on $P(v, e)$. Since γ is invisible from the endpoint v of e_i, there must be an edge $e'' \in E(P(v, e)) \setminus \{e\}$ such that $\varphi(e'') > \gamma$. At any edge on $P(v, e)$ traversed after e'', we have $r_{max} \geq \varphi(e'')$. Since e is traversed after e'' on $P(v, e)$, at e, we have $r_{max} \geq \varphi(e'')$ and $\varphi(e'') > \gamma$. Therefore at e, we have $r_{max} > \varphi(e)$. So $\varphi(e) = \gamma$ cannot be added to L. So L cannot contain any invisible rank. $\mathcal{Q.E.D.}$

Next we show that the rank chosen for e_i, the current new edge, does not violate the edge-ranking property in $T(e_i)$.

Lemma 2 The rank α properly ranks the newly arrived edge $e_i \in E(T)$ at the ith iteration.

Proof. For a contradiction, assume that α does not properly rank e_i. So $T(e_i)$ will contain a path $P(e', e'')$ for some $e', e'' \in E(T(e_i))$ such that $e_i \in E(P(e', e''))$, $\varphi(e') = \varphi(e'') = \gamma$, and $\varphi(e) \leq \gamma$ for all edges $e \in E(P(e', e''))$. Let $e_i = (u, v)$. Then $\varphi(e_i) = \alpha \leq \gamma$ and $\varphi(e')$ is visible from u and $\varphi(e'')$ is visible from v under φ. So $count(L(e_i), \gamma) \geq 2$. Since by *On_line_Edge_Ranking_Tree* $\alpha \notin L(e_i)$, $\alpha = \gamma$ is not possible. Thus $\gamma \geq \alpha + 1$. But $count(L(e_i), \beta) \leq 1$, for each β satisfying $\alpha + 1 \leq \beta \leq max\{L(e_i)\}$, according to *On_line_Edge_Ranking_Tree*. So α properly ranks e_i. $\mathcal{Q.E.D.}$

Proof of Theorem 1: For each execution of the RankEdge procedure, the visibility list is constructed by the BVL procedure using a recursive depth first search (DFS) traversal. For a tree, the complexity of DFS is $O(|E|) = O(|V|) = O(n)$, so it takes $O(n)$ time to build the visibility list. Since the size of the visibility list is $O(|E(T(e_i))|) = O(|E|) = O(|V|) = O(n)$, searching in $L(e_i)$ to find the rank α in Line 4 of procedure RankEdge takes $O(n)$ time. So we can say that the RankEdge procedure takes time $O(n)$. Since the procedure RankEdge is called for each newly arrived edge, and the m edges $e_1, e_2 \ldots, e_m$ arrive one at a time, the total time complexity of *On_line_Edge_Ranking_Tree* is $\sum_{i=1}^{m} O(n) = O(m) \cdot O(n) = O(n) \cdot O(n) = O(n^2)$, since for a tree $m = n - 1$. $\mathcal{Q.E.D.}$

4 Conclusion

In this paper, for the first time, we have presented an $O(n^2)$ time on-line algorithm for ranking the edges of a tree. Since only a subset of the input graph(tree) is available in each iteration, and an assigned rank cannot be changed later, an on-line edge-ranking algorithm cannot guarantee optimality. Therefore we use a greedy strategy for our on-line algorithm to find the least possible rank for each new edge. Experimental simulation results have shown that the algorithm properly ranks the edges of a tree in quadratic time. It is interesting to note that the corresponding problem on vertices, that is, the on-line vertex-ranking problem for trees can be solved in $O(n^3)$ time [7]. Rahman *et al.* improved the run-time by showing that the vertices of a tree can be ranked in $O(n^2)$ time [11]. Thus the on-line edge-ranking problem, which is more complex than the vertex counterpart, runs in the same time complexity.

References

[1] E. Bruoth, and M. Horňák, "On-line ranking number for cycles and paths", *Discussiones Mathematicae, Graph Theory*, vol. 19, pp. 175-197, 1999,.

[2] A. V. Iyer, H. D. Ratliff, and G. Vijayan, "Optimal node ranking of trees", *Information Processing Letters*, vol. 28, pp. 225-229, 1998.

[3] A. V. Iyer, H. D. Ratliff, and G. Vijayan, "On an edge-ranking problem of trees and graphs", *Discrete Applied Mathematics*, vol. 30, pp. 43-52, 1991.

[4] M. A. Kashem, and M. Z. Rahman, "An optimal parallel algorithm for c-vertex-ranking of trees", *Information Processing Letters*, vol. 92, pp. 179-184, 2004.

[5] M. A. Kashem, X. Zhou, and T. Nishizeki, "Algorithms for generalized edge-rankings of partial k-trees with bounded maximum degree", *Proceedings of the 1st International Conference on Computer and Information Technology (ICCIT)*, pp. 45-51, 1998.

[6] M. A. Kashem, X. Zhou, and T. Nishizeki, "Algorithms for generalized vertex-rankings of partial k-trees", *Theoretical Computer Science*, vol. 240, pp. 407-427, 2000.

[7] C. Lee, and J. S. Juan "On-line ranking algorithms for trees", *Proceedings of the International Conference on Foundations of Computer Science, Monte Carlo Resort, Las Vegas, USA*, pp. 46-51, 2005.

[8] C. Lee, and J. S. Juan, "Parallel algorithm for on-line ranking in trees", *Proceedings of the 22nd Workshop on Combinatorial Mathematics and Computational Theory, National Cheng Kung University, Tainan, Taiwan*, pp. 151-156, 2005.

[9] T. W. Lam, and F. L. Yue, "Edge ranking of graphs is hard", *Discrete Applied Mathematics*, vol. 85, pp. 71-86, 1998.

[10] T. W. Lam, and F. L. Yue, "Optimal edge ranking of trees in linear time", *Algorithmica*, vol. 30, pp. 12-33, 2001.

[11] M. R. Rahman, M. E. Haque, M. Islam, and M. A. Kashem, "On-line algorithms for vertex-rankings of graphs", *Proceedings of the International Conference on Information and Communication Technology (ICICT 2007)*, pp. 22-26, 2007.

[12] A. A. Schäffer, "Optimal node ranking of trees in linear time", *Information Processing Letters*, vol. 33, pp. 91-96, 1989.

[13] I. Schiermeyer, Zs. Tuza, and M. Voigt, "On-line rankings of graphs", *Discrete Mathematics*, vol. 212, pp. 141-147, 2000.

[14] P. de la Torre, R. Greenlaw, and A. A. Schäffer, "Optimal edge ranking of trees in polynomial time", *Algorithmica*, vol. 13, pp. 592-618, 1995.

[15] X. Zhou, M. A. Kashem, and T. Nishizeki, "Generalized edge-rankings of trees", *The Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Fundamentals of Electronics, Communications and Computer Science*, vol. 81-A-2, pp. 310-320, 1998.

[16] X. Zhou, N. Nagai, and T. Nishizeki, "Generalized vertex-rankings of trees", *Information processing Letters*, vol. 56, pp. 321-328, 1995.