Efficient Algorithms for Rankings of Graphs(グラフのランキングを求める効率的アルゴリズム)

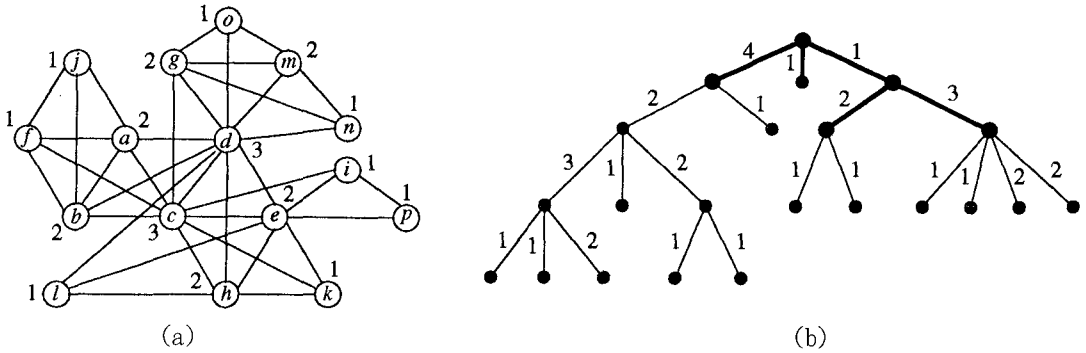
著者	MD.ABUL KASHEM MIA
号	79
発行年	1997
URL	http://hdl.handle.net/10097/12766

氏　名（本　籍）	MD. ABUL KASHEM MIA（モハマド アブル カシェム ミア）　（バングラディシュ）
学位の種類	博　士（情報科学）
学位記番号	情　博　第　79　号
学位授与年月日	平成10年3月25日
学位授与の要件	学位規則第4条第1項該当
研究科，専攻	東北大学大学院情報科学研究科（博士課程）システム情報科学専攻
学位論文題目	Efficient Algorithms for Rankings of Graphs （グラフのランキングを求める効率的アルゴリズム）
論文審査委員	（主査） 東北大学教授　西関　隆夫　　東北大学教授　白鳥　則郎 東北大学教授　阿曽　弘具（工学研究科）　　東北大学助教授　中野　眞一（工学研究科）

論文内容要旨

1 Introduction

An ordinary *vertex-ranking* of a graph G is a labeling (ranking) of the vertices of G with positive integers such that every path between any two vertices with the same label i contains a vertex with label $j > i$. Clearly a vertex-labeling is a vertex-ranking if and only if, for any label i, deletion of all vertices with labels $> i$ leaves connected components, each having at most one vertex with label i. A natural generalization of an ordinary vertex-ranking is the c-vertex-ranking [4]. For a positive integer c, a *c-vertex-ranking* of a graph G is a labeling of the vertices 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. Clearly an ordinary vertex-ranking is a 1-vertex-ranking. A c-vertex-ranking of G using the minimum number of ranks is called an *optimal c-vertex-ranking* of G. The *c-vertex-ranking problem* is to find an optimal c-vertex-ranking of a given graph. The problem is NP-hard in general. Bodlaender *et al.* presented a polynomial-time sequential algorithm to solve the ordinary vertex-ranking problem for partial k-trees [2]. Zhou *et al.* have obtained a linear-time sequential algorithm to solve the c-vertex-ranking problem for trees [4]. Figure 1(a) depicts an optimal 2-vertex-ranking of a graph G using three ranks, where ranks are drawn next to the circles. The c-vertex-ranking problem has an application to the parallel Cholesky factorization of matrices. The c-vertex-ranking problem of a graph G is equivalent to finding a c-vertex-separator tree of G having the minimum height.



(a)　　(b)

Figure 1: (a) An optimal 2-vertex-ranking of a graph G, and (b) an optimal 2-edge-ranking of a tree T.

The c-edge-ranking problem is defined analogously as for the c-vertex-ranking problem. For a positive integer c, a *c-edge-ranking* of a graph G is a labeling of the edges of G with integers such that, for any label i, deletion of all edges with labels $> i$ leaves connected components, each having at most c edges with label i. A c-edge-ranking of G using the minimum number of ranks is called an *optimal c-edge-ranking* of G. The *c-edge-ranking problem* is to find an optimal c-edge-ranking of a given graph G. Figure 1(b) depicts an optimal 2-edge-ranking of a tree T using four ranks, where the ranks are drawn next to the edges. The problem of finding an optimal c-edge-ranking of a graph G has applications in scheduling the parallel assembly of a complex multi-part product from its components, where the vertices of G correspond to the basic components and the edges correspond to assembly operations to be performed between the components. The c-edge-ranking problem of a graph G is also equivalent to finding a c-edge-separator tree of G having the minimum height. A c-edge-separator tree corresponds to a parallel computation scheme having the minimum computation time.

2 Preliminaries

In this chapter we define some terms. Let $G = (V,E)$ denote a graph with vertex set V and edge set E. We denote by n the number of vertices in G.

A natural generalization of ordinary trees is the so-called k-trees. The class of *k-trees* is defined recursively as follows:

(a) A complete graph with k vertices is a k-tree.

(b) If $G = (V,E)$ is a k-tree and k vertices $v_1, v_2, \cdots, v_k$ induce a complete subgraph of G, then $G' = (V \cup \{w\}, E \cup \{(v_i,w) \mid 1 \leq i \leq k\})$ is a k-tree, where w is a new vertex not contained in G.

(c) All k-trees can be formed with rules (a) and (b).

A graph is called a *partial k-tree* if it is a subgraph of a k-tree. In this thesis we assume that k is a fixed integer. The graph in Fig. 1(a) is a partial 3-tree.

A *tree-decomposition* of a graph $G = (V,E)$ is a pair (T,S), where $T = (V_T, E_T)$ is a tree and $S = \{X_x \mid x \in V_T\}$ is a collection of subsets of V satisflyng the following three conditions:

- $\cup_{x \in v_t} X_x = V$;
- for every edge $e = (v,w) \in E$, there exists a node $x \in V_T$ with $v,w \in X_x$; and
- for all $x,y,z \in V_T$, if node y lies on the path from node x to node z in T, then $X_x \cap X_z \subseteq X_y$.

The *width of a tree-decomposition* (T,S) is $\max_{x \in V_T} |X_x| - 1$. The *treewidth of a graph* G is the minimum width of a tree-decomposition of G, taken over all possible tree-decompositions of G. It is known that every graph G with treewidth $\leq k$ is a partial k-tree, and conversely, that every partial k-tree G has a tree-decomposition (T,S) with width $\leq k$ [1].

For any fixed integer k, determining whether the treewidth of a graph G is at most k and finding a binary tree-decomposition can be done in $O(n)$ sequential time [1]. On the other hand, for any fixed integer k, one can compute a binary tree-decomposition of G with height $O(\log_2 n)$ and width at most $3k+2$ in $O(\log_2^2 n)$ parallel time using $O(n)$ operations on the EREW PRAM[3].

3 Vertex-Rankings of Partial k-Trees

This chapter deals with the generalized vertex-ranking problem on partial k-trees. One of the main results of this chapter is the following theorem.

Theorem 3.1 *For any positive integer c and any bounded integer k, an optimal c-vertex-ranking of a partial k-tree G with n vertices can be found in* $n^{O(k^2)}$ *time.*

We first give an outline of a sequential algorithm for finding an optimal c-vertex-ranking of a partial k-tree G. Let (T,S) be a binary tree-decomposition of G with width $\leq k$. To each node x of T we associate a subgraph $G_x=(V_x,E_x)$ of G, where $V_x=\cup\{X_y \mid y = x$ or y is a descendant of x in $T\}$, and $E_x=\{(v,w)\in E \mid v,w\in V_x\}$. Thus G is associated with the root of T. The minimum number of ranks for c-vertex-ranking of a partial k-tree is $O(k \log_{c+1} n)$. For a given positive integer l, we decide whether G has a c-vertex-ranking $\phi\ \varphi$ using l ranks. We use dynamic programming and bottom-up tree computation on the binary tree T: for each node x of T from leaves to the root, we construct all(equivalence classes of) c-vertex-ranking of G_x from those of two subgraphs G_y and G_z associated with the children y and z of x. We then determine the minimum value of l such that G has a c-vertex-ranking φ using l ranks, and find an optimal c-vertex-ranking of G.

The time-complexity of a dynamic programming algorithm mainly depends on the size of the table, where each entry in the table represents an equivalence class. Therefore, we find a suitable equivalence class for which the table has a polynomial size. Before defining the equivalence class, we need to define some terms.

Let φ be a vertex-labeling of $G=(V,E)$ with positive integers. The label (rank) of a vertex $v\in V$ is denoted by $\varphi(v)$. A vertex $u\in V$ is said to be visible from a vertex $v\in V$ under φ in G if G has a path P from u to v every vertex of which has a rank $\leq \varphi(u)$. The rank $\varphi(u)$ of u is also said to be *visible* from v under φ in G if the vertex u is so. In Fig. 1(a) the vertices o,g,m,c and d are visible from o, but the vertex a is not visible from o.

Let $R=\{1,2,\ldots,l\}$ be the set of ranks. Let x be a node in T, and let $\varphi: V_x\to R$ be a vertex-labeling of the subgraph $G_x=(V_x,E_x)$. For an integer $i\in R$, we denote by *count*(φ,v,i) *the number of vertices ranked by* i and visible from $v\in X_x$ under φ in G_x. If φ is a c-vertex-ranking of G_x, then *count*$(\varphi,v,i)\leq c$ for any vertex $v\in X_x$ and any integer $i\in R$. We next define a *count-list* $L(\varphi,v)$ and a *list-set* $L(\varphi)$ as follows:

$L(\varphi,v)=\{(i,count(\varphi,v,i)) \mid$ rank $i\in R$ is visible from v under φ in $G_x\}$; and

$L(\varphi)=\{L(\varphi,v) \mid v\in X_x\}$.

For a vertex-labeling φ of G_x, define a function $\lambda_\varphi: X_x\times X_x\to R\cup\{0,\infty\}$ as follows:

$$\lambda_\varphi(v,w)=\min\{\lambda \mid G_x \text{ has a path } P \text{ from } v\in X_x \text{ to } w\in X_x \text{ such that } \varphi(u)\leq\lambda \text{ for each internal vertex } u \text{ of } P\}.$$

Let $\lambda_\varphi(v,w)=0$ if $(v,w)\in E_x$ or $v=w$, and let $\lambda_\varphi(v,w)=\infty$ if G_x has no path from v to w. We finally define a pair $R(\varphi)$ as follows:

$$R(\varphi)=(L(\varphi),\ \lambda_\varphi).$$

We call such pair $R(\varphi)$ the vector of φ on node x. $R(\varphi)$ is called a *feasible* vector if the vertex-labeling φ is a c-vertex-ranking of G_x.

A feasible vector $R(\varphi)$ of φ on x can be seen as an equivalence class of extensible c-vertex-rankings of G_x. Since $l=O(k\log_{c+1}n)$ and $|X_x|\leq k+1$, the total number of distinct feasible verctors on node x is at most $(c+1)^{O(k^2\log_{c+1}n)}\cdot(k\log_{c+1}n)^{k^2}=n^{O(k^2)}$. Computing each equivalence class on each node can be done in $O(n\log n)$ time, since the number of edges of a partial k-tree is $O(n)$ and the number of ranks is $O(\log n)$. Since the number of nodes of the tree-decomposition is $O(n)$ and the number of equivalence classes is $n^{O(k^2)}$, deciding whether a partial k-tree has a c-ranking using l ranks can be done in $n^{O(k^2)}$. $O(n^2\log n)=n^{O(k^2)}$time. Then determining the minimum value of l and finding an optimal c-vertex-ranking of a partial k-tree can be done in $n^{O(k^2)}$time.

Another main result of this chapter is the following theorem on our parallel algorithm for finding an optimal c-vertex-ranking of a partial k-tree.

Theorem 3.2 *Let G be a paratial k-tree of n vertices given by its tree-decomposition with height $O(\log_2 n)$ and width $\leq 3k+2$. Then an optimal c-vertex-ranking of G can be found in $O(\log_2 n)$ parallel time using $n^{O(k^2)}$operations on the common CRCW PRAM for any positive integer c and any bounded integer k.*

4 Edge-Rankings of Trees

The main result of this chapter is the following theorem.

Theorem 4.1 *For any positive integer c an optimal c-edge-ranking of a tree T can be found in time $O(n^2 \log_2 \Delta)$, where n is the number of vertices in T and Δ is the maximum degree of T.*

We give an outline of an algorithm to find an optimal c-edge-ranking of a tree T in time $O(n^2 \log \Delta)$. Since it is not so simple to find an optimal c-edge-ranking, we focus on specific types of optimal c-edge-rankings. Before formally defining them,we need to define some terms.

Let φ be an edge-labeling of a tree $T=(V,E)$ with positive integers. The label (rank) of an edge $e \in E$ is represented by $\varphi(e)$. One may assume without loss of generality that φ uses consecutive integers starting from 1 as the ranks. An edge $e \in E$ is said to be *visible* from a vertex $v \in V$ under φ in T if every edge in the path from e to v has a rank $\leq \varphi(e)$. In Fig. 1(b) all visible edges from the root are drawn in thick lines. An edge-labeling φ of a tree T is a c-edge-ranking if and only if no more than c edges of the same rank are visible from each vertex of T under φ. We define the *list* $L(\varphi)$ of a c-edge-ranking φ of tree T to be a list containing the ranks of all edges visible from the root, that is,

$$L(\varphi) = \{\varphi(e) \mid e \in E \text{ is visible from the root}\}.$$

The ranks in the list $L(\varphi)$ are sorted in non-increasing order.

We are now ready to define notions of optimality. A *critical* c-edge-ranking φ of tree T is defined to be a c-edge-ranking with the lexicographically smallest list $L(\varphi)$. Every critical c-edge-ranking φ is optimal, because all edges of the largest rank are visible and hence the topmost rank in $L(\varphi)$ is equal to the number of ranks used by φ. The optimal c-edge-ranking φ depicted in Fig. 1(b) is indeed critical. The maximal subtree of T rooted at a vertex v is denoted by $T(v)$. We define a c-edge-ranking φ of tree T to be *supercritical* if the c-edge-ranking is critical for every subtree $T(v)$ of T. Thus a supercritical c-edge-ranking is critical and hence optimal. The c-edge-ranking φ depicted in Fig. 1(b) is indeed supercritical. Since any supercritical c-edge-ranking is optimal, it suffices to give an algorithm for finding a supercritical c-edge-ranking of T.

We now give an outline of an algorithm to find a supercritical c-edge-ranking of a tree T. Our algorithm uses the technique of "bottom-up tree computation." That is, it repeats the following operation for each internal vertex v of a tree T from leaves to the root: constructs a supercritical c-edge-ranking φ of subtree $T(v)$ from those φ_1, $\varphi_2, \cdots, \varphi_d$ of the subtrees $T(w_1), T(w_2), \cdots, T(w_d)$ rooted at the children $w_1, w_2, \cdots, w_d$ of v. We decide the d ranks of the edges $e_i=(v,w)$, $1 \leq i \leq d$, in non-increasing order so that the c-edge-ranking extended from the supercritical c-edge-rankings of the subtrees $T(w_i)$, $1 \leq i \leq d$, is supercritical for $T(v)$. Let L_i, $1 \leq i \leq d$, be the list of the supercritical c-edge-ranking $T(w_i)$ which has been obtained. Since the largest rank among the d ranks of the edges e_i, $1 \leq i \leq d$, is unique, we first decide the largest rank β. Once β is decided, we label with β the edge e_j for which the ranks in L_j hidden by β is lexicographhlly largest. We then delete the subtree $T(w_j)$ and the edge e_j, and continue the same operations for the remaining smaller tree and find the $d-1$ remaining ranks. We can decide the rank of an edge in time $O(n \log \Delta)$, where Δ is the maximum degree of T. Repeating the same operations for $n-1$ edges, we can find a supercritical c-edge-ranking of T in time $O(n^2 \log \Delta)$.

5 Edge-Rankings of Partial κ-Trees with Bounded Degrees

In this chapter we first give a polynomial-time sequential algorithm to solve the c-edge-ranking problem for any partial k-tree with bounded degrees and any positive integer c. We next give a parallel algorithm for the c-edge-ranking problem. Our algorithms are similar to our sequential and parallel algorithms for the c-vertex-ranking of a partial k-tree. The main results of this chapter are the following theorems.

Theorem 5.1 *For any positive integer c and any bounded integer k, an optimal c-edge-ranking of a partial k-tree G with n vertices can be found in time* $n^{O(\Delta k^2)}$, *where* Δ *is the maximum degree of G.*

Theorem 5.2 *Let G be a partial k-tree of n vertices given by its tree-decomposition with height* $O(\log_2 n)$ *and width* $\leq 3k+2$. *Then an optimal c-edge-ranking of G can be found in* $O(\log_2 n)$ *parallel time using* $n^{O(\Delta k^2)}$ *operations on the common CRCW PRAM for any positive integer c and any bounded integer k.*

6 Conclusion

This thesis deals with the generalized vertex-ranking problem and the generalized edge-ranking problem. We newly define generalized edge-ranking problem on graphs and give an efficient algorithm for solving the problem on trees. We present efficient sequential and parallel algorithms for solving the generalized vertex-ranking problem and the generalized edge-ranking problem on partial k-trees. We also obtain upper bounds on the c-vertex-ranking number and the c-edge-ranking number of partial k-trees.

References

[1] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, *SIAM Journal on Computing*, **25**(1996), pp.1305-1317.

[2] H.L. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, H. Müller, and Zs. Tuza, Rankings of graphs, *Lecture Notes in Computer Science, Springer-Verlag*, **903** (1994), pp.292-304.

[3] H.L. Bodlaender and T. Hagerup, Parallel algorithms with optimal speedup for bounded treewidth, *Lecture Notes in Computer Science, Springer-Verlag*, **944** (1995), pp.268-279.

[4] X. Zhou, H. Nagai, and T. Nishizeki, Generalized vertex-rankings of trees, *Information Processing Letters*, **56** (1995), pp.321-328

審 査 結 果 の 要 旨

並列計算機のプロセッサへのタスクの割当問題など多くのスケジューリング問題は，ある制約条件を満足するようにグラフの点や辺にランクを付け，しかも用いるランクの種類をできるだけ少なくする，いわゆるランク付け問題として定式化できる。このようなランク付け問題はNP-完全であり，一般のグラフに対してこの問題を効率よく解くアルゴリズムは存在しそうにないが，グラフのクラスを少し限定すれば効率のよいアルゴリズムが存在すると予想される。しかし，どのようなグラフのクラスに限定すればよいかは知られていなかった。

著者は新しいランク付け問題を定式化するとともに，新しい手法やデータ構造を導入し，木およびその一般化である部分k木に対して，その問題を解く逐次アルゴリズムおよび並列アルゴリズムを設計し，その効率を理論的に解析した。本論文はこれらの成果をとりまとめたものであり，全編6章からなる。

第1章は序論である。

第2章では，グラフの基本的な概念やランク付けの一般的な性質を示している。

第3章では，部分k木の点ランク付け問題を解く効率のよい逐次アルゴリズムと並列アルゴリズムを与えている。入力グラフの点数をnとしたとき，この逐次アルゴリズムの計算時間はnの多項式である。また並列アルゴリズムの計算時間は$O(\log n)$であり，用いるプロセッサ数はnの多項式である。部分k木の点ランク付け問題がいわゆるNCに属することを示した点で，この成果は高く評価できる。

第4章では，木の辺ランク付け問題を解く効率のよい逐次アルゴリズムを与えている。木の最大次数をΔとしたとき，この逐次アルゴリズムの計算時間は$O(n^2 \log \Delta)$であり，このアルゴリズムは極めて高速で，実用上も有用である。

第5章では，部分k木の辺ランク付け問題を解く逐次アルゴリズムと並列アルゴリズムを与えている。部分k木の最大次数が定数であるとき，この逐次アルゴリズムの計算時間はnの多項式である。また並列アルゴリズムの計算時間は$O(\log n)$であり，用いるプロセッサ数はnの多項式である。これらは重要な成果である。

第6章は，結論である。

以上要するに，本論文は木および部分k木のランク付け問題を解く逐次アルゴリズムと並列アルゴリズムを与え，その効率を理論的に解析したものであり，計算機科学，特にアルゴリズム理論の発展に寄与するところが少なくない。

よって，本論文は博士（情報科学）の学位論文として合格と認める。