# A Linear Time Algorithm for Finding a $k$-Tree-Core

Akiyoshi SHIOURA * and Takeaki UNO *

(November, 1994)

## 1 Introduction

Let $T = (V, E)$ be a tree with $n$ vertices. For two vertices $u$ and $v$, we define the distance $d(u, v)$ as the number of edges on the unique path between $u$ and $v$, and $d(u, S) = \min_{v \in S} d(u, v)$ for $S \subseteq V$. Given a positive integer $k$, we consider the problem of finding a $k$-leaf-subtree (subtree which contains exactly $k$ leaves) $S$ which minimizes $D(S) = \sum_{v \in V} d(v, S)$, the sum of the distances from all vertices to $S$. Such a $k$-leaf-subtree is called a $k$-tree-core of $T$.

The problem of finding a $k$-tree-core is one of several types of location problems for a single facility on a tree which minimizes the sum of the distance. The oldest, posed by Hakimi [2], is the problem of finding a vertex called a "node median" or a "distance centroid", which minimizes the sum of distance. This may be extended naturally to paths , and a path which minimizes the total distance is called a "core" or "path median", and linear time algorithms for finding a core have been proposed by Morgan and Slater [5], and Peng et al. [6]. Minieka and Patel [3] added a constraint on the length of a path, and defined a "core of length $l$" as a path of length $l$ which minimizes the total distance. This problem is extended to a tree-shaped facility in [4]. On the other hand, the problem of finding a $k$-tree-core, which we treat here, adds a different constraint namely such that the subtree must have exactly $k$ leaves. This problem was first considered by Peng et al.[6] who gave two algorithms for finding a $k$-tree-core whose time complexities are $O(kn)$ and $O(n \log n)$. The latter algorithm can find $k$-tree-cores for all $k$ in $O(n \log n)$.

In this paper, we propose a linear time algorithm for finding a $k$-tree-core. Our algorithm is a modified version of the $O(kn)$-algorithm of Peng et al. and is very simple while theirs are little complecated. It first finds a core in linear time, then finds $k-2$ paths needed to construct a $k$-tree-core, and adds them. We show that these added paths have some special properties, which allows us to find them in $O(n)$ time. Peng et al. showed similar properties, but our lemmas and proofs are simple and clear. Furthermore, with a slight modification, our algorithm can find $k$-tree-cores for all $k$ in linear time.

We also consider a $k$-tree-core in weighted tree. By using our algorithm, we can find a $k$-tree-core in linear time, but it takes $O(n \log n)$ time to find $k$-tree-cores for all $k$. We show that $\Omega(n \log n)$ is the lower bound for solving the latter problem, and that therefore our algorithm is optimal for this.

*Department of Information Sciences, Tokyo Institute of Technology, 2-12-1 Oh-okayama, Meguro-ku, Tokyo 152, Japan. shioura@is.titech.ac.jp , uno@is.titech.ac.jp

In Section 2, we give some notation and definitions, and show some basic properties about distance. In Section 3, we prove some useful properties for our algorithm, and propose a linear time algorithm. Finally we discuss $k$-tree-cores in a weighted tree in section 4.

## 2 Preliminaries

Let $T = (V, E)$ be a tree. $P_{uv}$ denotes the unique path which connects two vertices $u$ and $v$. The *distance* between two vertices $u$ and $v$ is defined by the number of edges in the path $P_{uv}$, and is denoted by $d(u, v)$. For a vertex $u$ and a subtree $S$ in $T$, the *distance* between $v$ and $S$ is defined by $d(v, S) = \min_{u\in S}\{d(u, v)\}$.

Here we define some measure of "centrality" of subtrees in $T$. For a vertex $v$, the *distance of* $v$, denoted by $D(v)$, is defined as the sum of the distances between $u$ and $v$ for all vertices $u \in V$, i.e., $D(v) = \Sigma_{u\in V} d(u, v)$. Similarly, for a subtree $S$ in $T$, $D(S) = \Sigma_{u\in V} d(u, S)$ is called the *distance of* $S$.

A *core* of a tree $T$ is a path which minimizes the distance $D(P)$ among all paths $P$ in $T$. A *$k$-tree-core* is a subtree which minimizes the distance $D(S)$ among all subtrees $S$ containing exactly $k$ leaves. We can see that a core is a 2-tree-core. It is easily shown that each leaf of $k$-tree-core is also a leaf of $T$. Note that a $k$-tree-core is not always uniquely defined.

A vertex $v \notin S$ is *adjacent* to a subtree $S$ if there exists an edge $(u, v)$ with $u \in S$. For a vertex $r \in T$ and a vertex $v \neq r$, we consider 'rooting' $T$ at $r$. We denote the subtree (of this rooted tree) rooted at $v$ as $T_r(v)$. More generally, for a subtree $S$ and a vertex $v \notin S$, let $T_S(v)$ be the subtree in $T$ induced by the vertex-set $V_S(v) = \{x | P_{vx} \cap S = \emptyset \text{ and } d(x, S) \geq d(v, S)\}$. If we regard $T$ as a tree 'rooted' at $S$, $T_S(v)$ can be seen as a subtree rooted at $v$.

If a subtree $S$ becomes larger, the distance $D(S)$ decreases strictly. So, we consider decreasing $D(S)$ by adding a path $P$ to a subtree $S$. The following equation holds for the decrease of the distance by addition of a path to a subtree.

**Property 2.1** *Let $P$ be a path in $T$ and $v$ be one of endpoints of $P$. Let $S$ be any subtree of $T$ which intersects $P$ only at the vertex $v$. Then,*

$$D(S) - D(S \cup P) = D(v) - D(P)$$

**Proof:**

$$\begin{aligned}
D(S) - D(S \cup P) &= \sum_{u\in V}\{d(u, S) - d(u, S \cup P)\} \\
&= \sum_{u\in T_S(v)}\{d(u, v) - d(u, P)\} + \sum_{u\notin T_S(v)}\{d(u, S) - d(u, S)\} \\
&= \sum_{u\in T_S(v)}\{d(u, v) - d(u, P)\} + \sum_{u\notin T_S(v)}\{d(u, v) - d(u, v)\} \\
&= \sum_{u\in V}\{d(u, v) - d(u, P)\}
\end{aligned}$$

∎

This means that for any subtree $S$ which intersects $P$ at only one endpoint $v$, $D(S) - D(S \cup P)$ has the same value. We call this value the *distance saving* of $v$ and $P$, and denote it by $DS(v, P)$. The distance saving has the following property.

**Property 2.2** *Let $v$ and $w$ be two distinct vertices, and $v'$ be the vertex in $P_{vw}$ adjacent to $v$. Then,*

$$DS(v, P_{vw}) = DS(v', P_{v'w}) + |T_v(v')|$$

**Proof:**

$$\begin{aligned} DS(v, P_{vw}) &= D(v) - D(P_{vw}) \\ &= D(v) - \{D(P_{vv'}) - DS(v', P_{v'w})\} \\ &= DS(v, P_{vv'}) + DS(v', P_{v'w}) \\ &= DS(v', P_{v'w}) + |T_v(v')| \end{aligned}$$

▮

By this property, we can compute $DS(v, P_{vw})$ from $DS(v', P_{v'w})$ immediately. It is one of the keys of our algorithm.

# 3 An algorithm for finding a $k$-tree-core

In this section, we propose an algorithm for finding a $k$-tree-core. We assume that $k$ is less than the number of leaves in $T$.

Our algorithm is based on the $O(kn)$ algorithm by Peng et al.[6]. Their algorithm finds a core at first, and adds $k-2$ paths iteratively. It takes $O(n)$ time for finding each path, hence $O(kn)$ time is required for finding all $k-2$ paths. Our algorithm also finds a core in the first step. After that, we construct a set of paths, and by adding $k-2$ elements selected from this set to the core, we get a $k$-tree-core. We can execute this step in $O(n)$, thus a linear time algorithm for finding a $k$-tree-core may be realized. We show some lemmas, which were first proved by Peng et al.[6].

**Lemma 3.1** [6] *For any $k$-tree-core $S \neq T$, there exists a $(k+1)$-tree-core $S'$ such that $S \subset S'$.* ▮

By using this lemma, we can construct a $(k+1)$-tree-core from a given $k$-tree-core $S_k$ by adding a path which minimizes the distance. Here we consider the path which maximizes $DS(v, P)$. For a subtree $S$ in $T$ and a vertex $v \notin S$, let $u$ be the vertex adjacent to $v$ such that $d(u, S) = d(v, S) - 1$. When a path $P$ maximizes $DS(u, P)$ among all paths $P_{uw}$ with $w \in T_S(v)$, we call $P$ the *local rooted core* of $v$ with respect to $S$ and denote it by $LRC(v, S)$, The next property is implied by Property 2.2.

**Property 3.2** *Let $S$ be a subtree in $T$ and $v$ be a vertex which maximizes $DS(LRC(v, S))$ among all vertices not in $S$. Then, $v$ is adjacent to $S$.* ▮

From the definition of local-rooted-core, the previous lemma can be rewritten as follows.

**Corollary 3.3** *For any $k$-tree-core $S$, let $P$ be a local-rooted-core $LRC(v, S)$ which maximizes the distance saving among all vertices $v$ adjacent to $S$. Then, $S \cup P$ is a $(k+1)$-tree-core.*

▮

Now, we consider how to find a local-rooted-core $LRC(v, S)$. Suppose $v$ is not a leaf of $T$. Let $u$ be a vertex which is adjacent to $v$ and satisfies $d(u, S) = d(v, S) - 1$. Let $\{v_1, \cdots, v_r\}$ be vertices which are adjacent to $v$ and satisfy $d(v_i, S) = d(v, S) + 1$. Such vertices surely exist because $v$ is not a leaf. From the definition of local-rooted-cores, the following relation is implied.

$$\begin{aligned} DS(LRC(v,S)) &= \max\{DS(u,P_{uw})|w \in T_S(v)\} \\ &= \max\{DS(v,P_{vw})|w \in T_S(v)\} + |T_S(v)| \\ &= \max[\max_{1\le i\le r}\{\max\{DS(v,P_{vw})|w \in T_S(v_i)\}\},\ DS(v,P_{vv})] + |T_S(v)| \\ &= \max_{1\le i\le r} i\{\max\{DS(v,P_{vw})|w \in T_S(v_i)\}\} + |T_S(v)| \\ &= \max_{1\le i\le r}\{DS(v,LRC(v_i,S))\} + |T_S(v)| \end{aligned}$$

By using this relation, we can compute a local-rooted-core $LRC(v,S)$ recursively.
**Algorithm** *Find_LRC*$(v,S,T)$ (Find $LRC(v,S)$ for $v \notin S$ and subtree $S \subset T$.)

**Step 0:** Let $u$ be the vertex which is adjacent to $v$ and satisfies $d(u,S) = d(v,S) - 1$.

**Step 1:** If $v$ is a leaf of $T$ then return the path $P_{uv}$. Stop.

**Step 2:** If $v$ is not a leaf of $T$, then let $\{v_1,\cdots,v_r\}$ be vertices which are adjacent to $v$ and which satisfy $d(v_i,S) = d(v,S) + 1$. Find a local-rooted-core $LRC(v_i,S)$ for each vertex $v_i$.

**Step 3:** Choose the path $P^*$ with the largest value of distance saving.

**Step 4:** Return the path $P^* \cup \{(u,v)\}$. Stop.

Moreover, we can also compute all local-rooted-cores $LRC(x,S)$ for $x \in T_S(v)$ simultaneously as byproducts. Now we consider the time complexity of this algorithm. Let $Time(v)$ be the time required to compute $LRC(v,S)$. Then,

$$\begin{aligned} Time(v) &= \sum_{i=1}^{r} Time(v_i) + O(\deg(v)) \\ &= O(\sum_{u\in T_S(v)} \deg(u)) \\ &= O(|T_S(v)|) \end{aligned}$$

Hence, it takes $O(n)$ time to compute local-rooted-cores $LRC(v,S)$ for all vertices $v \notin S$. The algorithm of Peng et al. iteratively computes local-rooted-cores $k$ times, and takes $O(kn)$ time to find a $k$-tree-core.

In our algorithm for finding a $k$-tree-core, we compute a core $C$ first. Then we make a set of local-rooted-cores $LS$ by using the algorithm *Find_LRC*$(v,S,T)$. Here we consider local-rooted-cores produced by the algorithm *Find_LRC*$(v,S,T)$. When we find $LRC(v,S)$, we find local-rooted-cores $LRC(v_i,S)$ for $i = 1,\cdots,r$. One of them $LRC(v_{i^*},S)$ is included in $LRC(v,S)$, and the others intersect $LRC(v,S)$ at only one vertex $u$. We define $LS$ as the set of maximal local-rooted-cores, i.e., $LS = \{LRC(v,C) \mid LRC(v,C) \not\subset LRC(w,C), \forall w \neq v\}$

We also define the *body* of $LRC(v,C)$ as the sub-path $LRC(v,C) \cap T_C(v)$.

**Property 3.4**
*1. Each vertex $v \in T\backslash C$ is contained in exactly one body of a local-rooted-core $L \in LS$.*
*2. For any local-rooted-core $L \in LS$ and any vertex $v \notin L$ adjacent to the body of $L$, a local-rooted-core of $v$ is contained in $LS$.*

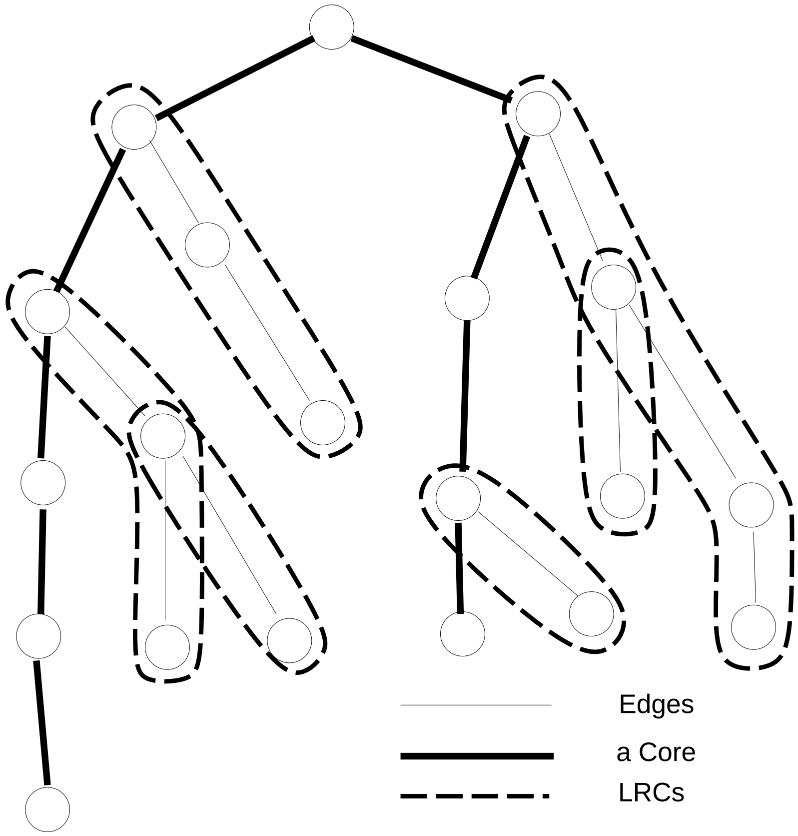


Figure 1: The set of local-rooted-cores $LS$

**Proof:** Clearly, any vertex $w \in T \backslash C$ is contained in at least one local-rooted-core of $LS$. If $w$ is contained in two bodies of $LRC(v_1, C)$ and $LRC(v_2, C)$, then $LRC(v_1, C) \subset LRC(v_2, C)$ or $LRC(v_2, C) \subset LRC(v_1, C)$. Hence, by definition of $LS$, $LS$ contain the maximal local-rooted-core which contains $LRC(v_1, C)$ (and $LRC(v_2, C)$).

For a local-rooted-core $L \in LS$, let $v \notin L$ be any vertex adjacent to the body of $L \in LS$, and $L' \in LS$ be the unique local-rooted-core which contains $v$ in its body. Suppose $L' \neq LRC(v, C)$ and let $u \in L$ be the vertex which is adjacent to $v$. Then $u$ is also contained in the body of $L'$, which is a contradiction. Therefore, $L' = LRC(v, C)$. ∎

The next lemma ensures the correctness of our algorithm.

**Lemma 3.5**

*Let $L_i$ be the element in $LS$ with $i$-th largest value of distance saving. Then,* $S_k = \bigcup_{i=1}^{k-2} L_i \cup C$ *is a $k$-tree-core.*

**Proof:** If $k = 2$ then this statement holds obviously. So, for $k > 2$, we assume that $S_{k-1}$ is a $(k-1)$-tree-core and show that $S_k$ is $k$-tree-core.

From Corollary 3.3, $S_k$ is a $k$-tree-core if and only if $L_{k-2}$ maximizes the distance saving among all local-rooted-cores $LRC(v, S)$ such that $v$ is adjacent to $S_{k-1}$. For any vertex $v$, if $v$ is adjacent to core $C$, or $v$ is adjacent the body of some local-rooted-core $LRC(x, C)$ and not in $LRC(x, C)$, then $LS$ has a local-rooted-core $LRC(v, C)$ in it. Therefore, for each vertex $v$ adjacent to $S_{k-1}$, $LRC(v,C) \in LS \setminus \{L_i \mid i = 1,2,\cdots,k-3\}$, and if $LRC(v,C) \in LS \setminus \{L_i \mid i = 1,2,\cdots,k-3\}$ then $v \notin S_{k-1}$. From Property 2.2, $L_{k-2}$ maximizes the distance saving among all local-rooted-core $LRC(v,S)$ such that $v$ is adjacent to $S_{k-1}$, since $L_{k-2}$ has the largest value of distance saving in $LS \setminus \{L_i \mid i = 1,2,\cdots,k-3\}$. Hence, $S_k$ is a $k$-tree-core. ∎

Now, we formulate our algorithm.

**Algorithm** *Find_k-tree-core*$(k,T)$

**Step 1:** Find a core $C$.

**Step 2:** Compute $LS$.

**Step 3:** Sort elements in $LS$ in the decreasing order of the distance saving by using radix sort.

**Step 4:** Output $C$ and the $k-2$ largest elements in $LS$.

**Theorem 3.6**
*Algorithm Find_k-tree-core*$(k,T)$ *outputs a* $k$*-tree-core of a tree* $T$ *in* $O(n)$ *time and uses* $O(n)$ *space.*

**Proof:** Steps 1 and 2 can be done in $O(n)$ time. In Step3, we sort all elements of $LS$. Radix sort takes only $O(d(n+e))$ time and $O(n+e)$ space if each number is a positive integer less than $e^d$, hence Step3 can be done in $O(n)$ time, because the distance saving of any path is a positive integer less than $n^2$. The size of the output is at most the size of a given tree $T$, and Step 4 takes $O(n)$ time. Hence, this algorithm runs in $O(n)$ time.

In Steps 1, 2, and 4, the memory requirement is proportional to the size of a given graph. By the above argument about radix sort, we use only $O(n)$ space when we sort all elements in $LS$. Therefore, the space complexity is $O(n)$. ∎

From lemma 3.5, the differences between a $k$-tree-core and a $(k-1)$-tree-core is the local-rooted-core $L_{k-2}$. Therefore, in the previous algorithm, if we output all local-rooted-cores $L_1, L_2, \cdots$ instead of outputting only $L_1, \cdots, L_{k-2}$, we can reconstruct all $k$-tree-cores for $k \geq 2$. That is, we can find all $k$-tree-cores for any $k$ in linear time.

## 4 $k$-tree-cores in weighted graphs

In this section, we discuss the problem of finding a $k$-tree-core in weighted tree. We consider a tree $T=(V,E)$ such that each edge $e \in E$ has an arbitrary positive length $l(e)$ and each vertex $v \in V$ has an arbitrary positive weight $w(v)$. We define the distance $d(u,v)$ between vertices $u$ and $v$ by the length of the path $P_{uv}$, i.e., $d(u,v) = \sum_{e \in P_{uv}} l(e)$. The distance between one vertex $v$ and one subtree $S$ is defined by $d(v,S) = \min_{u \in S} d(u,v)$. The distance of a subtree $S$ is defined as the value $D(S) = \sum_{v \in V} w(v)d(v,S)$. By using this distance, we can define a $k$-tree-core similarly to the unweighted case. Thus we can find a $k$-tree-core in the same manner except for the sorting of the elements of $LS$. In a weighted graph, we cannot use radix sort. However, this is no problem because we do not have to sort all elements in $LS$ to find the $k-2$ largest elements. In fact, the $k$-best selection algorithm suffices, and we can find a $k$-tree-core in $O(n)$ time.

Next, we consider finding $k$-tree-cores for all $k$. In this case, we must sort all elements in $LS$ and it takes $O(n \log n)$ time to find $k$-tree-cores by using our algorithm. Here we

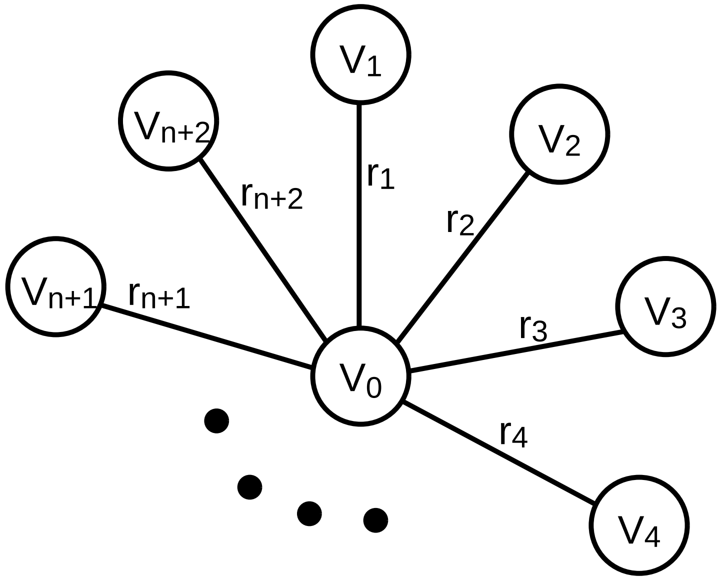


Figure 2: star-shaped graph $G$

show that this is equal to the lower bound of time complexity to output each $k$-tree-core of a weighted tree for all $k$, by reducing the sorting problem to it. It is well-known that the problem of sorting $n$ numbers requires $\Omega(n \log n)$ time. We exhibit the fact that the sorting problem is transformable in linear time to the problem of outputting each $k$-tree-core, and prove the lower bound of our problem. For a given sequence of real numbers $\{r_1, ..., r_n\}$ , we consider a star-shaped tree graph $G$ which has vertices $\{v_0, ..., v_{n+2}\}$ and edges $(v_i, v_0)$ for $i = 1, 2, \cdots, n+2$. We assume that all numbers $r_i (i = 1, \cdots, n)$ are distinct. In this graph, vertices $\{v_1, ..., v_{n+2}\}$ are leaves. We define the weight of each edge $(v_i, v_0)$ as $r_i$ for $i \leq n$, and as a sufficient large value, e.g., $\max_j\{r_j\} + 1$ for $i > n$ (see Figure 2). Clearly, a core $C$ of the graph $G$ is the path from $v_{n+1}$ to $v_{n+2}$. The path consisting of only one edge $(v_i, v_0)$ is a local-rooted-core of vertex $v_i \notin C$. Let $S_k$ be a $k$-tree-core of $G$. We can easily see that $S_k$ contains edges in $\{(v_i, v_0) \mid i = 1, \cdots, n+2\}$ with $k$-th largest weight, and $S_{k+1} \setminus S_k$ contains the edge $(v_0, v_i)$ with $k$-th largest weight. If we output each $k$-tree-core $S_k$ for all $k$ by outputting differences $S_{k+1} \setminus S_k$, we can sort numbers $\{r_1, \cdots, r_n\}$. This means that it requires $\Omega(n \log n)$ time to find differences between $S_k$ and $S_{k+1}$ for all $k$.

## Acknowledgment

We are greatly indebted to Prof. Akihisa Tamura of the University of Electro-Communications and Dr. Yoshiko Ikebe of Tokyo Institute of Technology for valuable comments and help on this manuscript.

## References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *“’Introduction to Algorithms”*, The MIT Press, Massachusetts, 1990.

[2] S. L. Hakimi, *"Optimum Location of Switching Centers and the Absolute Centers and Medians of a Graph"*, Oper. Res. 12 (1964), 450 - 459.

[3] E. Minieka and N. H. Patel, *"On Finding the Core of a Tree with a Specified Length"*, J. Algorithms 4 (1983), 345 - 352.

[4] E. Minieka, *"The Optimal Location of a Path or Tree in a Tree Network"*, Networks 15, No. 3 (1985), 309 - 321.

[5] C. A. Morgan and P. J. Slater, *"A Linear Algorithm for a Core of a Tree"*, J. Algorithms 1 (1980), 247 - 258.

[6] S. Peng, A. B. Stephens, and Y. Yesha, *"Algorithms for a Core and k-Tree Core of a Tree"*, J. Algorithms 15 (1993), 143 - 159.

[7] A. B. Stephens, Y. Yesha, and K. Humenik, *"Optimal Allocation for Partially Replicated Database Systems on Tree-Based Networks"*, Working paper.