Iranian Journal of Numerical Analysis and Optimization
Vol. 9, No. 1, (2019), pp 93–104
DOI:10.22067/ijnao.v9i1.65852

On the finding 2-(k,l)-core of a tree with arbitrary real weight

S.M. Ashkezari and J. Fathali*

Abstract

Let $T = (V, E)$ be a tree with $\mid V \mid = n$. A 2-(k, l)-core of T is two subtrees with at most k leaves and with a diameter of at most l, which the sum of the distances from all vertices to these subtrees is minimized. In this paper, we first investigate the problem of finding 2-(k, l)-core on an unweighted tree and show that there exists a solution that none of (k, l)-cores is a vertex. Also in the case that the sum of the weights of vertices is negative, we show that one of (k, l)-cores is a single vertex. Then an algorithm for finding the 2-(k, l)-core of a tree with the pos/neg weight is presented.

Keywords: Core; Facility location; Median subtree; Semi-obnoxious.

1 Introduction

Location theory is one of the important fields in operations research. In the classical location theory, we want to find the optimal location of a set of single points, as the facilities, on a network. However, in many real applications, the facility to be located is too large to be modeled as a point, so the extensive facilities, which have the form of a path or tree, are considered in many researches.

A tree T is given. A core of a tree is a path of T, so that the sum of the weighted distances from all vertices to this path is minimized. Morgan and Slater [5] and Becker [1] presented linear time algorithms for finding a core

*Corresponding author
Received 6 July 2017; revised 25 July 2018; accepted 1 September 2018
S.M. Ashkezari
Faculty of Mathematical Sciences, Shahrood University of Technology, University Blvd., Shahrood, Iran. email: samane.motevalli@gmail.com

J. Fathali
Faculty of Mathematical Sciences, Shahrood University of Technology, University Blvd., Shahrood, Iran. email: fathali@shahroodut.ac.ir

of a tree, where the weights of all vertices are nonnegative. Zaferanieh and Fathali [16] considered the problem of finding the core of a general network and presented an ant colony and simulated annealing algorithms for solving this problem. Rahbari et al. [10] presented a hybrid genetic and an ant colony algorithm for the path center problem on general networks. Motevalli and Fathali [6] presented a linear time algorithm for finding the core of a tree with interval weights.

A 2-core of a tree is a set of two paths minimizing the sum of the distances of all vertices of the tree from these two paths. Becker and perl [3] considered both cases of disjoint paths and intersecting paths for a tree. Wang [13] presented a linear time algorithm for disjoint paths case.

Some authors deal with the case that there is a constraint on the length of path, so that the length can be at most l (see, e.g., [2,4,8]). Peng et al. [9] considered the problem of finding a subtree containing exactly k leaves, called k-tree core, such that it minimizes the sum of the distances of all vertices to this tree. A linear time algorithm is given for this problem by Shioura and Uno [11]. Wang [12] and Wang et al. [14] considered the parallel algorithms for finding the k-tree core of a tree.

An special case of k-tree core is (k, l)-core, which the diameter of k-tree is at most l. Becker et al. [2] presented an efficient algorithm for finding a (k, l)-core of a tree with time complexity of $O(n^2 logn)$. Their algorithm is started by the tree T and constructs new rooted trees where the maximum length of a path is at most l. Then, for each new tree, apply a greedy-type procedure to find a subtree containing the root with at most k leaves and which minimizes the sum of the distances. The problem of finding the (k, l)-core of a tree also has been considered in [15].

Recently the semi-obnoxious location problems have found an increasing interest. Zaferanieh and Fathali [17] considered the problem of finding a core of a tree with pos/neg weights. The problem of finding a (k, l)-core of a tree with pos/neg weights is considered by Motevalli et al. [7]. They proved that, when the sum of the weights of vertices is negative, the (k, l)-core must be a single vertex. They also presented an algorithm with time complexity of $O(n^2 logn)$ for finding the (k, l)-core of a tree with the pos/neg weight, which is in fact a modification of the one proposed by Becker et al. [2]. Recently Zhou et al. [19] presented polynomial time algorithms for finding 2-core of a tree with pos/neg weights.

In this paper, we consider the problem of finding 2-(k, l)-core of T, where the weights of vertices in T can be any positive or negative numbers. A 2-(k, l)-core of tree is two disjoint subtrees T_1 and T_2 of T that each of them has at most k leaves and diameter of at most l, so that the sum of the distances from all vertices to these subtrees is minimized. Note that if these two subtrees are intersecting, then the problem converts to a (k, l)-core problem that has been solved in [7].

In what follows, we define the problem in Section 2. Then we consider the unweighted tree in Section 3 and show that there exists a solution that

any of 2-(k,l)-core trees is not a vertex. In Section 3, also we show that when the sum of the weights of vertices is negative, one of 2-(k,l)-cores is a single vertex. At last, we propose an algorithm for finding the 2-(k,l)-core of a tree with the pos/neg weight in Section 4.

2 Problem Formulation

Let $T = (V, E)$ be a tree with $|V| = n$. Let $w(v_i)$, for simplicity w_i, be the weight of vertex $v_i \in V$ and let $a(i,j)$ be the length of edge (i,j). Also let $d(v_i, v_j)$ be the length of path from v_i to v_j; then the length of shortest path between subtree T_1 and vertex v is given by

$$d(v, T_1) = min_{u \in T_1}\ d(u, v).$$

For every path P, we show the length of P by $L(P)$, and for every subtree T' of T, we show the weight of tree T' by $w(T')$, that is, $w(T') = \sum_{v_i \in T'} w_i$. The diameter $d_{T'}$ of any tree T' is the maximum distance between two vertices of T'. Any path whose length equals $d_{T'}$, is called a diameter path of T'.

A (k,l)-core of T is a subtree $T_1 = (V_1, E_1)$ of T with at most k leaves and with a diameter of at most l, so that the sum of weighted distances from all vertices to T_1 is minimized, that is,

$$minF(T_1) = \sum_{v_i \in V \setminus V_1} w_i d(v_i, T_1).$$

Also a 2-(k,l)-core of T is a set of two subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ with at most k leaves and with a diameter of at most l, so that the following function is minimized:

$$F(T_1, T_2) = \sum_{v_i \in V \setminus (V_1 \cup V_2)} w_i d(v_i, T_1, T_2),$$

where $d(v_i, T_1, T_2)$ is the minimum distances from the vertex v_i to T_1 and T_2, that is,

$$d(v_i, T_1, T_2) = min\{d(v_i, T_1), d(v_i, T_2)\}.$$

The applications of 2-(k,l)-core is the same as (k,l)-core. It can be applied to design of communication networks such as railroad lines, high ways, pipelines, and transit routes. For more details on semi-obnoxious (k,l)-core and its application to real world, we refer the reader to [7] and [17].

3 The properties for special cases

In this section, we investigate some properties of 2-(k,l)-core in special cases. The basic idea of solving 2-(k,l)-core is the edge deletion method, which is tried to find an optimal solution by removing any edges and find a (k,l)-core in each obtained subtree. In the following theorems, we use this idea to show the properties of problem.

First, we consider the unweighted tree where all edges and vertices have the same positive weight. The following theorem shows that none of two subtrees of 2-(k,l)-core on an unweighted tree with $d_T > 2$ is a single vertex.

Theorem 1. *Let $T = (V, E)$ be an unweighted tree. Then, for the case $l > 0$ and $d_T > l+1$, there is a set of two subtrees T_1' and T_2', which are 2-(k,l)-core of T, that non of them is a single vertex.*

Proof. First, we show that a leaf cannot be one of 2-(k,l)-core of T. Let q be an edge between an inner vertex and a leaf v_i of T. By removing q, two subtrees T_1 and $T_2 = v_i$ are obtained. Let T_1' and $T_2' = v_i$ be the (k,l)-core of T_1 and T_2, respectively, and let $v_j \in T_1$ be the adjacent vertex to v_i. If $v_j \notin T_1'$, then by adding v_j to T_2', the objective function will not be increased. In the case when $v_j \in T_1'$, let $v_r \in T_1'$ be a non leaf adjacent vertex to v_j and let $T_1'(v_j)$ be a subtree of T_1' containing v_j obtained by deleting the edge (v_j, v_r). Then by adding all vertices in $T_1'(v_j)$ to T_2' and deleting them from T_1', the objective function does not increase. Note that since $d_T > l+1$, then there exists a non leaf vertex $v_r \in T_1'$ adjacent to v_j.

In the case that one of (k,l)-cores is an inner vertex, obviously, we can reduce the objective function by extending the core toward a leaf. □

Note that Theorem 1 is not true for the case $d_T \leq 2$. A counter example, for this case is the star graph, which is a tree that have just one vertex with degree more than one.

Now consider the case $w(T) < 0$. In this case, according to Theorems 2 and 3, the solution of (k,l)-core is a single vertex that may be a leaf. The proof of these theorems can be found in [7].

Theorem 2. *Let T be a tree with $w(T) < 0$. Then the (k,l)-core is the 1-median of T and vice versa.*

For any edge $(u, v) \in E$, suppose that T_{uv} is the subtree of T obtained by deleting edge (u, v), in which $v \in T_{uv}$.

Theorem 3. *Let T be a tree with $w(T) < 0$. Then one of the following two cases holds:*

1. *The (k,l)-core is a leaf.*
2. *The (k,l)-core is an inner vertex u, so that $w(u) \geq 0$ and $w(T_{uv}) \leq 0$ for each vertex v adjacent to u.*

Corollary 1. *Let $T = (V, E)$ be a tree with $w(T) < 0$. There is a solution for a $2-(k,l)$-core problem that one of (k,l)-cores is a single vertex.*

Proof. Let T_1' and T_2' be the $2-(k,l)$-core of T. Also let T_1 and T_2 be two subtrees of T containing all vertices which assigned to T_1' and T_2', respectively. So T_1' and T_2' is (k,l)-core of T_1 and T_2, respectively. Since $w(T) < 0$, then $w(T_1) < 0$ or $w(T_2) < 0$. Therefore by Theorem 2, T_1' or T_2' is a vertex. □

Theorem 4. *Let $T = (V, E)$ be a tree with $w(T) \geq 0$, let T_1 be a subtree of T with $d_{T_1} < l$, and let the number of leaves of T_1 be less than k. If $w(T \setminus T_1) > 0$, then T_1 is not a (k,l)-core of T.*

Proof. Let $T_1^c = T \setminus T_1$. Since $w(T_1^c) > 0$, there exists a component T_2 of T, which is obtained by removing T_1, so that $w(T_2) > 0$. Let $u \in T_2$ be the adjacent vertex to T_1 and $T_3 = T_1 \cup \{u\}$. Then

$$\begin{aligned} F(T_1) &= \sum_{v_i \in T_1^c \setminus T_2} w_i d(v_i, T_1) + \sum_{v_i \in T_2} w_i d(v_i, T_1) \\ &= \sum_{v_i \in T_1^c \setminus T_2} w_i d(v_i, T_1) + \sum_{v_i \in T_2} w_i (d(v_i, u) + d(u, T_1)) \\ &= F(T_3) + d(u, T_1) w(T_2) > F(T_3). \end{aligned}$$

Therefore T_1 is not the (k,l)-core of T. □

Zhou et al. [19] proved that the midpoint of two disjoint facility paths is a vertex of the tree. This property holds for 2-(k,l)-core case and can be written as the following theorem.

Theorem 5. *If two disjoint subtrees T_1 and T_2 are a 2-(k,l)-core of T, then the midpoint of the path between T_1 and T_2 is a vertex of the tree.*

4 Algorithm

As mentioned before to find a 2-(k,l)-core of a tree, we delete an edge of T and divide T to two subtrees T_1 and T_2. Then find the (k,l)-core of each of T_1 and T_2. This procedure is repeated for each edge of T and the best pair of (k,l)-cores is found. To find any (k,l)-core of T_1 and T_2, we can use the algorithm of Motevalli et al. [7]. The complexity of their algorithm is $O(n^2 logn)$, therefor the complexity of finding 2-(k,l)-core is $O(n^3 logn)$.

However for the case $w(T) < 0$, according to Lemma 3, the (k,l)-core is a vertex, which is either a leaf or an inner vertex u, so that $w(u) \geq 0$ and $w(T_{uv}) \leq 0$ for each vertex v adjacent to u. Also in the case $w(T) > 0$, we can use Theorem 4.

Therefore we can present the following algorithm.

Algorithm
Input: a tree T with the pos/neg weight
Output: a 2-(k,l)-core S^* of T and its objective function d^*
begin
$d^* := +\infty$
for each subtree T_1 and T_2 obtained from T by removing each edge $e \in E$ **do**
SUBTREE(T_i) for $i = 1, 2$
end

Where procedure SUBTREE(T) find (k,l)-core of tree T as follows:

Proedure $SUBTREE(T')$
Input: a subtree $T' = (V', E')$ of T with $\mid V' \mid = n'$ and the best current objective function d^*
Output: if the best subtree in T' has an objective function less than the previous value of d^*, the best subtree S^* in T' , having at most k leaves and with a diameter of at most l and its objective function as the new value of d^*
begin
if T' consists of one vertex **then**
$S' = T'$ and let $d(S')$ be its objective function
else
find a central vertex v of T'
if $w(T') < 0$, v is an inner vertex, $w(v) \geq 0$ and for each vertex u adjacent to v, $w(T_{vu}) \leq 0$
then v is the (k,l)-core, let $S' = v$ and set $d(S')$ as its objective function
else let $S' :=$ **BEST-TREE**(T', v)
if $d(S') < d^*$ and $W(T \setminus S') < 0$
$d^* := d(S')$
$S^* := S'$
for each subtree T^i obtained from T' by removing v **do**
SUBTREE(T^i)
end

In this procedure, a central vertex v of the tree T is the centroid of the corresponding unweighted tree of T, that is, a vertex minimizes the maximum number of vertices of the subtrees obtained by removing it. The procedure BEST-TREE(T', v), which is presented by Beker et al. [2], finds the best subtree S' containing v in T'^v having at most k leaves and a diameter of at most l.
Procedure $BEST - TREE(T', v)$
Input: a subtree $T' = (V', E')$ of T with n' vertices rooted at the central vertex v

Output: the best subtree S' containing v in T'^v having at most k leaves and a diameter of at most l and its objective function $d(S')$

begin
find the paths starting from v with the length at most l
for each path P starting from v with the length at most l
do
prune the tree T'^v and let $\hat{T'^v}$ be the new tree (see Prune)
find the distance savings of the paths from v to each vertex $u \neq v$ in $\hat{T'^v}$
if $deg(v) \geq 2$ **then**
find the path P'
find $P'' = P \cup P'$
else
$P'' = P$
find the set LS with all paths having the length no more than l and
containing P'' that their objective function are better than P''
if $\hat{T'^v}$ has more than k leaves, **then**
in LS select the $k-2$ paths to be added to P''
let S' be the best subtree containing v and
let $d(S')$ be its objective function
else
select all the paths in LS
let $S' = \hat{T'^v}$ and let $d(S')$ be its objective function
end for
end

The following notations are used in procedure BEST-TREE.

Let P_{uv} be the path between vertices u and v in the tree T and let T^u be the tree T rooted at u. We also denote by T^u_v the rooted subtree of T^u containing v and all the descendants of v. For each vertex u, let $f(u)$ be its father. The distance saving $sav(v, P_{vu})$ obtained by adding P_{vu} to the root v in T^u_v, is given by

$$sav(v, P_{vu}) = sav(v, P_{vf(u)}) + a(f(u), u)sum_c(u)$$

where if $v = f(u)$, then $sav(v, P_{vf(u)}) = sav(v, v) = 0$ and $sum_c(u)$ is calculated by

$$sum_c(v) = \begin{cases} w(v) & \text{if } v \text{ is a leaf of} T_c, \\ w(v) + \sum_{\text{u is a child of } v} sum_c(u) & otherwise, \end{cases} \quad (1)$$

where T_c is the selected subtree of T as the (k,l)-core in the current iteration. Let the path $P = P_{vu}$ be given and let B be the set of children of v. We denote by $T_{\bar{b}}^v$, $\bar{b} \in B$, the subtree in which P lies. In the algorithm, the tree T^v is pruned as follows:

1. Prune the paths that belong to subtrees T_b^v, with $b \in B \setminus \{\bar{b}\}$, in order to obtain paths with the length at most $min\{l - L(P), L(P)\}$.
2. For the paths that lie in the same subtree as $P = P_{vu}$ (i.e., in $T_{\bar{b}}^v$), prune those paths P_{xw}, with $x \in P$, $x \neq v$ and $w \in T_{\bar{b}}^v \setminus P$, such that $L(P_{xw}) > min\{L(P) - L(P_{vx}), l - L(P_{xu})\}$.

The pruned tree is called $\hat{T^v}$, and the weight of tree $\hat{T^v}$ is calculated by the following function:

$$w'(u) = \begin{cases} w(u) & \text{for each vertex } u \text{ of } \hat{T^v} \text{that is not a leaf,} \\ sum_v(u)a(u, f(u)) & \text{for each vertex } u \text{ that is a leaf of } \hat{T^v}. \end{cases} \quad (2)$$

In the rest of this section, we use the algorithm for a different case of the total weight of a tree.

Example 1. Consider the tree shown in Figure 1. The length of each edge is written on it. Also the weights of vertices are shown in Table 1. The total weight of tree is $w(T) = 3 > 0$. We want to find the 2-(3,3)-core on this tree. Since $w(T) > 0$, according to the algorithm, we should delete an edge of T and obtain two subtrees T_1 and T_2. Let us delete $v_9 - v_{10}$ and consider $T_1 = v_{10}$ and $T_2 = T \setminus T_1$. Now we should find (3,3)-core of each subtree. Because T_1 has only one vertex (v_{10}), so its (3,3)-core is v_{10}. To find the (3,3)-core of T_2, we should find a central vertex of T_2. We start with v_5 as a central vertex. Then we should find all of the paths starting from v_5 with the length at most 3. This paths are shown in Figure 2.

Table 1: The weights of vertices of tree in Figure 1 for Example 1

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}
3	-2	1	-1	1	2	-3	2	2	-1	-2	1

Now for each pat, prune the tree T and continue steps of the algorithm. For example, consider the path number 8 in Figure 2, the new tree after prune is presented in Figure 3.

The distance savings of $p : v_5\_v_4$ and $p : v_5\_v_6$ are $sav(v_5, p_{v_5v_4}) = -2$ and $sav(v_5, p_{v_5v_6}) = +2$. Since $deg(v_5) = 4$, we consider $p' = v_5 - v_6$, so $p'' = p \cup p'$ that is shown in Figure 3. Also since $\hat{T'^v}$ has 2 leaves, so we select all the paths in LS; then $S' = \hat{T'^v}$ and $d(S') = 0$. After performing the above steps for each path, we consider subtrees obtained by removing v_5 one by one and

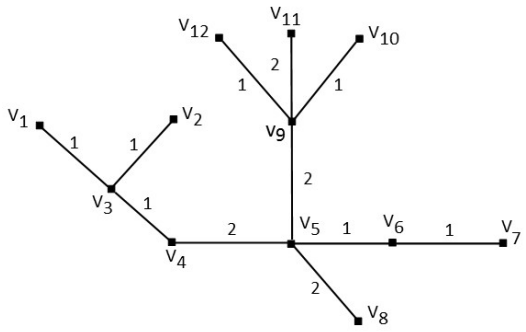

Figure 1: A tree with 12 vertices.

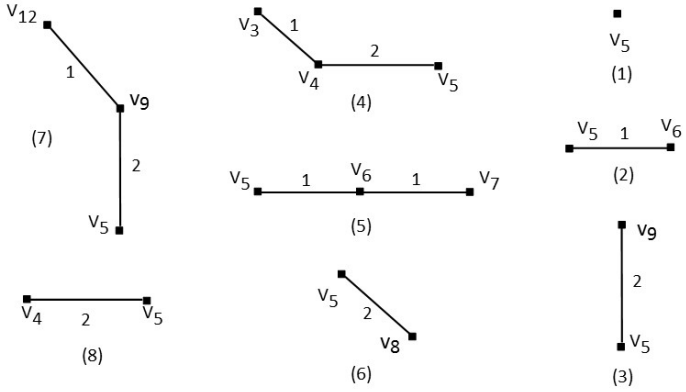

Figure 2: The paths starting from v_5 with the length at most 3.

find a central vertex for each subtree and continue above steps for each subtree. After that we delete other edges of tree such as $v_9 - v_{10}$ one by one and run all of the above steps after removing each edge. Finally 2-(3,3)-core of T is $T_1' = v_1$ and $T_2' = v_8$ that is obtained by removing of edge $v_1 - v_3$, and its objective function is equal to $F(T_1', T_2') = F(v_1) + F(v_8) = 0 + (-19) = -19$.

Example 2. Consider the tree depicted in Figure 4. The edge lengths are written on the edges. The weights of vertices are given in Table 2. The total weight of tree is $w(T) = -1 < 0$. We want to find the 2-$(3,4) - core$ on this tree. Since $w(T) < 0$, according to Theorem 2, the (k,l)-core is a vertex of T. The (k,l)-core of T is v_3, that is 1-median of T, too. The best solution is obtained by removing edge $v_6 - v_8$.Therefore 2-(k,l)-core is $T_1' = v_3$ and $T_2' = v_8$,

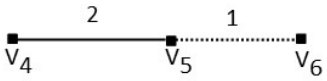

Figure 3: The path number 8 after prune.

and the objective function is $F(T_1', T_2') = F(T_1') + F'(T_2) = -25 + 0 = -25$.

Table 2: The weights of vertices of tree in Figure 1 for Example 1

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
-1	3	2	-2	1	-3	-2	1

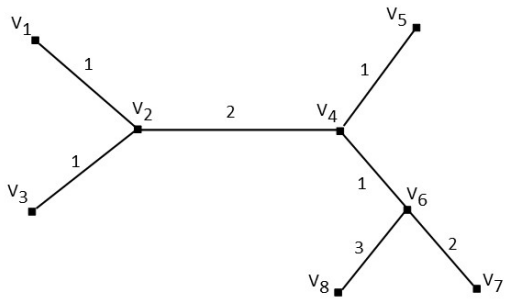

Figure 4: A tree with 8 vertices.

5 Summary and conclusion

In this paper, we considered the 2-(k, l)-core problem on a tree with positive and negative weights. We showed that, in the case when the sum of weights of tree is negative, the solution of 2-(k, l)-core is also 1-median. Some properties also stated for the case that the tree has the positive weight. Then a polynomial algorithm was presented to find the solution of this problem.

For a future research, it would be interesting to develop the algorithm on other special graphs such as extended stars, cactus graphs, interval graphs, and block graphs.

Acknowledgements

Authors are grateful to there anonymous referees and editor for their constructive comments.

References

1. Becker R.I. *Inductive algorithms on finite trees,* Quaest Math., 13, (1990), 165–181.

2. Becker R.I., Lari I., Storchi G. and Scozzari A. *Efficient algorithms for finding the (k, l)-core of tree networks,* Networks, 40, (2002), 208–215.

3. Becker R.I. and Perl Y. *Finding the two-core of a tree,* Discrete Applied Mathematics, 11, (1985), 103–113.

4. Minieka E. and Patel N.H. *On finding the core of a tree with a specified length,* J. Alg., 4, (1983), 345–352.

5. Morgan C.A. and Slater P.J. *A linear algorithm for a core of a tree,* Journal of Algorithms, 1, (1980), 247–258.

6. Motevalli S. and Fathali J. *A linear algorithm for finding core of weighted interval trees*, Journal of Operational Research and Its Applications, 13, (2016), 101–111.

7. Motevalli S., Fathali J. and Zaferanieh M. *An efficient algorithm for finding the semi-obnoxious (k,l)-core of a tree,* Journal of Mathematical Modeling, 3, (2015), 129–144.

8. Peng S. and Lo W. *Efficient algorithms for finding a core of a tree with a specified length,* Journal of Algorithms 20, (1996), 445–458.

9. Peng S., Stephens A.B. and Yesha Y. *Algorithms for a core and a k-tree core of a tree*, J. Alg., 15, (1993), 143–159.

10. Rahbari M., Fthali J. and Mortazavi R. *A hybrid algorithm for the path center problem,* Global Analysis and Discrete Mathematics, 1, (2016), 83–92.

11. Shioura A. and Uno T. *A linear time algorithm for finding a k-tree core,* J. Alg., 23, (1997), 281–290.

12. Wang B.F. *Finding a k-tree core and a k-tree center of a tree network in parallel,* IEEE Transactions on Parallel and Distributed Systems, 9, (1998), 186–191.

13. Wang B.F. *Finding a 2-core of a tree in linear time,* SIAM Journal on Discrete Mathematics, 15, (2002), 193–210.

14. Wang Y., Wang D.Q., Liu W. and Tian B.Y. *Efficient parallel algorithms for constructing a k-tree center and a k-tree core of a tree network,* Lecture Notes in Computer Science 3827, Springer-Verlag Berlin Heidelberg, (2005), 553–562.

15. Wang Y. and Wang Y. *Efficient algorithms for constructing a (k,l)-center and a (k,l)-core in a tree network,* Fourth International Conference on Innovative Computing, Information and Control, (ICICIC), 2009.

16. Zaferanieh M. and Fathali J. *Ant colony and simulated annealing algorithms for finding the core of a graph,* World Applied Science Journal, 7, (2009), 1335–1341.

17. Zaferanieh M. and Fathali J. *Finding a core of a tree with pos/neg weight,* Math. Meth. Oper. Res., 76, (2012), 147–160.

18. Zelinka B. *Medians and peripherians of trees,* Archvum Mathematicum, 4, (1968), 87–95.

19. Zhou J., Kang L. and Shan E. *Two paths location of a tree with positive or negative weights,* Theoretical Computer Science, 607, (2015), 296–305.

مساله پیدا کردن $(k,l)-2$–هسته روی درختهای با وزن حقیقی

سمانه متولی اشکذری و جعفر فتحعلی

دانشگاه صنعتی شاهرود، دانشکده علوم ریاضی

دریافت مقاله ۱۵ تیر ۱۳۹۶، دریافت مقاله اصلاح شده ۳ مرداد ۱۳۹۷، پذیرش مقاله ۱۰ شهریور ۱۳۹۷

چکیده :
فرض کنید $T=(V,E)$ درختی شامل n راس باشد. یک $(k,l)-2$–هسته از درخت T شامل دو زیر درخت مجزا از T می باشد که هر یک از آنها حداکثر دارای T برگ و حداکثر قطر l هستند، همچنین مجموع فاصله تمام رئوس درخت تا این دو زیردرخت کمینه است. در این مقاله در ابتدا به بررسی مساله پیدا کردن $(k,l)-2$–هسته روی درختی که در آن وزن تمام رئوس یکسان است می پردازیم و نشان می دهیم جوابی برای این مساله وجود دارد که هیچ یک از هسته ها یک راس تنها نیست. سپس الگوریتمی برای پیدا کردن $(k,l)-2$–هسته روی درختهایی که وزن رئوس آنها می تواند مثبت یا منفی باشد، ارائه می دهیم.

کلمات کلیدی : هسته درخت؛ مکانیابی نیمه-ناخوشایند؛ زیردرخت میانه.