Journal of Universal Computer Science, vol. 20, no. 14 (2014), 1951-1963
submitted: 2/1/14, accepted: 22/11/14, appeared: 1/12/14

A Self-stabilizing Algorithm for Locating the Center of Maximal Outerplanar Graphs

Michał Pańczyk
(Maria Curie-Skłodowska University in Lublin, Poland
mjpanczyk@gmail.com)

Halina Bielak
(Maria Curie-Skłodowska University in Lublin, Poland
hbiel@hektor.umcs.lublin.pl)

Abstract: Self-stabilizing algorithms model distributed systems, which automatically recover from transient failures in the state of the system.

The center of a graph comprises a set of vertices with minimum eccentricity. Farley and Proskurowski showed the linear time algorithm for locating the center of an outerplanar graph in the classical computing paradigm. The present paper investigates the self-stabilizing algorithm for finding the center of maximal outerplanar graphs, using a new approach with dual trees.

Key Words: self-stabilization, outerplanar graph, center of a graph
Category: G.2.2, C.2.4

1 Introduction

Let $G = (V, E)$ be a simple, connected graph with the vertex set V and edge set E. The *eccentricity* of the vertex v in the graph G is its distance to the farthest vertex in the graph G. The set of vertices with minimum eccentricity is called the *center* of the graph.

Locating centers of graphs, especially in distributed systems, has profound significance. It allows to minimize the cost of communicating all the nodes with one elected node from the center. This is useful when, for example, a control center is to be placed in a distributed network. There are several known algorithms for locating centers or medians of graphs. Specific topological properties of graphs can be employed to achieve algorithms with lower complexity or with a simpler notation. Farley [Farley 1982] gave a linear time algorithm for vertex centers in trees; also, Hedetniemi et al. [Hedetniemi et al. 1981] gave a linear time algorithm for center problems in trees; Goldman [Goldman 1972] and Kariv et al. [Kariv et al. 1979a, Kariv et al. 1979b] devised algorithms solving the center problem in networks. A lot of research has been realized related to centers of graphs [Laskar et al. 1983, Rosenthal et al. 1989, Chepoi 2012]. Self-stabilizing algorithm for centers and medians of trees was developed by [Bruell et al. 1999].

Distributed (but not self-stabilizing) algorithm for centers and medians was developed by [Korach et al. 1984]. Bielak and Pańczyk constructed a self-stabilizing algorithm for finding weighted centroid in trees [Bielak and Pańczyk 2012].

In this paper, we propose a self-stabilizing algorithm for locating the center of the maximal outerplanar graph. This problem in classical, sequential, non-distributed computing paradigm was solved by Farley and Proskurowski [Farley and Proskurowski 1980]. The structure of centers in maximal outerplanar graphs was researched by [Proskurowski 1980]. To our knowledge, there is no known self-stabilizing algorithm which employs the topological structure of maximal outerplanar graphs to find the center. Whereas a self-stabilizing algorithm for trees is devised by [Blair and Manne 2003].

The next section contains basic notations and definitions. The section 3 describes the computational model of self-stabilizing algorithms used in our paper. The section 4 contains the text of a classical algorithm for locating the center of maximal outerplanar graphs by [Farley and Proskurowski 1980]. In the section 5, we present our new self-stabilizing adaptation of the classical algorithm, which uses, to our knowledge, a new method based on the inside dual tree. It also uses a method of co-working nodes, which simulate a kind of virtual nodes of the dual tree. Conclusions, including further research problems, are described in the section 6.

2 Basic Notations and Definitions

Let us define the maximal outerplanar graphs as it was done in the above mentioned paper [Farley and Proskurowski 1980].

Definition 1. A maximal outerplanar graph is obtained by a planar triangularization of a plane polygon (see Fig. 1).

According to the definition we used, maximal outerplanar graphs are 2-connected and the minimal type of-such a graph is a triangle. All the vertices can be placed on the same face; however, they are usually placed on the exterior face [Harary 1972].

Definition 2. In the maximal outerplanar graph $G = (V, E)$ every edge $p = \{i, j\} \in E$ divides the set of all vertices apart from i and j into two distinct sets inducing connected subgraphs called sides.

One of the sides may be empty. This is the case when the dividing edge is a part of the exterior face of the graph. In fact, all the edges with one side empty form the unique Hamilton cycle in the maximal outerplanar graph.

Let us note that in a maximal outerplanar graph every two neighbors i and j have at most two common neighbors, say k and l, each of them belonging to distinct sides of the edge $\{i, j\}$. Thus, it is sufficient to represent the side of the edge $\{i, j\}$ by the unique vertex (say k or l) belonging to this side and adjacent to both i and j. We set $\emptyset$ as a representation of an empty side.

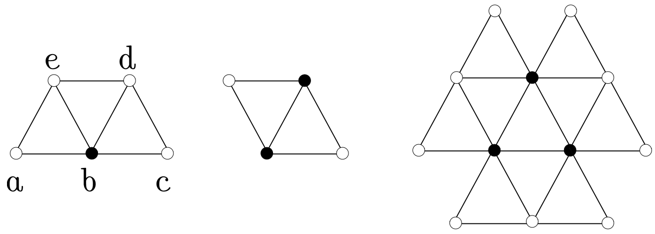

Figure 1: Examples of maximal outerplanar graphs with centers of one, two and three nodes.

Farley and Proskurowski introduced the notion of edge eccentricity.

Definition 3. Let us have the node i, its neighbor j and one of their common neighbors k ($\emptyset$ for the nonexistent one if applicable) in a graph G. For these three values we define $e(i, j, k)$ (the edge eccentricity) in the following manner:

– the absolute value of $e(i, j, k)$ is equal to the eccentricity of the vertex i in the subgraph of G induced by $S_k \cup \{i, j\}$, where S_k is the side of the edge $\{i, j\}$ containing the vertex k,

– the value $e(i, j, k)$ is negative iff all the vertices of $S_k \cup \{i, j\}$ at a distance $d = |e(i, j, k)|$ from i lie at the distance $d - 1$ from the vertex j.

The classical algorithm of Farley and Proskurowski computes the edge eccentricity for every edge recursively using the already computed values of eccentricity for an adjacent edge. It starts with outerface edges, for which 2 out of 4 defined edge eccentricities equal -1. We include an essential part of [Farley and Proskurowski 1980] in the section 4.

3 Computational Model

A notion of self-stabilizing algorithms on distributed systems was introduced by [Dijkstra 1974]. A survey in the topic can be found in the paper [Schneider 1993], and further details in the book by [Dolev 2000]. The notions from graph theory not defined in this paper can be found in the book by [Harary 1972].

A distributed self-stabilizing system consists of a set of processes (computing nodes) and communication links between them. Every node in the system runs the same algorithm and can change the state of local variables. These variables determine the *local state* of a node. Nodes can observe the state of variables in themselves and in their neighbor nodes. The state of all the nodes in the system determines the *global state*.

Every self-stabilizing algorithm should have a class of global states defined, so-called the *legitimate state*, for which the system is stable and in which no action can be taken by the algorithm itself. Every other global state is called *illegitimate* and for the algorithm to be correct there has to be some possibility to make a move if the state is illegitimate.

The aim of a self-stabilizing algorithm is to bring the system to the legitimate (desirable) state, either after some alteration (from outside of the system) of variables in the nodes has been done or after the system has been started.

The algorithm consists of a set of rules. A rule has the following form:

label
  If guard
  then assignment instructions
  where definitions of objects.

A *guard* is a predicate which can refer to variables in the node itself and its neighbors. The *where* clause is optional. We say that a rule is *active* if its guard is evaluated to be true. A node is *active* if it contains any active rule. If there is no active node in the graph, then the system is *stabilized*. We assume that active rules are triggered in an arbitrary order one by one. Time spent on running a rule is negligible compared to the time spent on transmitting a message between processes. Thus, our objective is to minimize the number of transmitted messages between nodes.

4 The Classical Algorithm

In order for the paper to be self-contained, we quote the algorithm for locating the center of maximal outerplanar graphs, basic lemmas and the figure (see Fig. 2) from [Farley and Proskurowski 1980].

Lemma 4. *Given an edge $p = (u, v)$ of a maximal outerplanar graph with a non-empty side S, let e_1, e_2 and r represent the values of $e(b, w, S_2)$, $e(b, v, S_2)$ and the eccentricity of u in the graph $S_2 \cup \{u, v, w\}$, respectively. Then the value of r is $-(1 + e_2)$ if $e_2 > 0$, and $|e_2|$ otherwise.*

Lemma 5. *Given an edge $p = (u, v)$ with a non-empty side S, let e_3 and d_1 represent the values $e(a, u, S_1)$ and $e(p, u, S)$, respectively. Let r be the eccentricity of u in the graph $S_2 \cup \{u, u, w\}$ as in Lemma 4. Then the value of d_1 is $|e_3|$ if $|e_3| \geq |r|$, and r otherwise.*

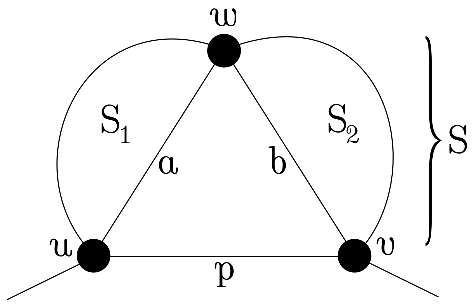

Figure 2: Illustration for lemmas 4, 5 and 6 from [Farley and Proskurowski 1980].

Lemma 6. *Given an edge $p = (u, v)$ with a non-empty side S, the eccentricity $d_2 = e(p, v, S)$ is $|e(b, v, S_2)|$ if $|e(b, v, S_2)| \geq |q|$, and q otherwise*[1]*, where q is $-(1 + e(a, u, S_1))$ if $e(a, u, S_1) > 0$, and $|e(a, u, S_1)|$ otherwise.*

In the paper [Farley and Proskurowski 1980] the correctness of the above lemmas was proved. Moreover, the authors applied them in their classical (non-distributed) paradigm algorithm presented below.

Algorithm 1: [Farley and Proskurowski 1980]

Given a maximal outerplanar graph M, calculate the eccentricities of its edges as follows:

1. For all edges $p = (u, v)$ on the Hamiltonian cycle of M, assign the value -1 to $e(p, x, \emptyset)$, where $x \in \{u, v\}$.
2. For each triangle (u, v, w) such that values $e(a, u, S_1)$, $e(a, w, S_1)$, $e(b, v, S_2)$, and $e(b, w, S_2)$ are defined, assign values of $e(p, u, S)$ and $e(p, v, S)$ according to the rules specified in Lemmas 4, 5, and 6.

5 The Self-Stabilizing Algorithm

In this section, we shall present a self-stabilizing adaptation of the non-distributed algorithm for locating the center of maximal outerplanar graphs. We will use the notion of inside (weak) dual graph [Fleischner et al. 1974] (see Fig. 3). Its vertices correspond to the inner regions of maximal outerplanar graphs, whereas

[1] In the original version, there was a small technical defect: was $|e(b, v, S_2)|$, should be q in this case.

the edges of an inside dual graph correspond to the inside edges (i.e. edges incident to two inner regions). The information about dual vertices will be stored in the three nearest base co-working vertices (real computing nodes), which will cooperate to simulate the existence of the dual vertex.

It is quite easy to show that the inside dual graph of a maximal outerplanar graph is acyclic and connected; therefore, it is a tree.

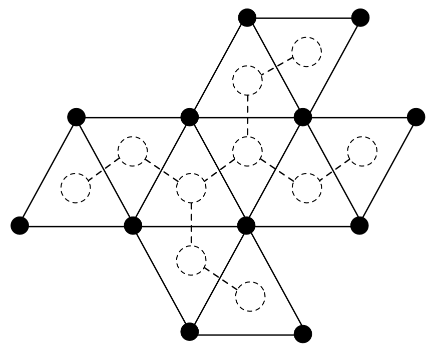

Figure 3: An example of a maximal outerplanar graph and its inside dual tree.

Below, we present the notation used in the algorithm. Next, the rules of the algorithm follow and then the presentation of the idea and intuition connected with the rules. At the end of this section, we present the proofs of the correctness of the algorithm.

In the algorithm, we will use the following notation:

$N(i)$ — the set of neighbor nodes of the node i.

$n(i)$ — a variable storing the set of neighbor nodes for the node i. Note that $n(i)$ is a variable, whose value may be incorrect at the beginning of the algorithm run, whereas $N(i)$ is a set which is determinable by the node i only, based on the topology of the network by looking at connections of node i; it can be computed only by the node i. Thus the $n(i)$ variable is set to allow a neighbor to determine other neighbors of the node i.

$c(i, j)$ — a variable (stored in the node i) which stores the set of common neighbors for nodes i and j,

$e(i, j, k)$ — a variable storing (in the node i) the edge eccentricity for the edge $\{i, j\}$ and the side containing the common neighbor k (of the nodes i and j; $k = \emptyset$ for an empty side),

$opp(i, j, k)$ — a variable storing (in the node i) the representation of the side opposite to k (against the edge $\{i, j\}$),

$v(i)$ — a variable storing (in the node i) the eccentricity of the vertex i; note that it is not the *edge* eccentricity, i.e. in a legitimate state: $v(i) = \max_k |e(i, j, k)|$ for any $j \in N(i)$,

$m(i, j, k)$ — a pair stored in the node i for inside dual vertex $\{i, j, k\}$. After stabilization, its first element is the eccentricity of the center nodes. The second element of the pair is the direction that the information about the eccentricity of the center comes from. If for the dual vertex $\{i, j, k\}$ the information comes from the region incident to the edge $\{i, j\}$, then the direction is equal to $opp(i, j, k)$. If the information about the center eccentricity comes originally from the dual vertex $\{i, j, k\}$, then we set the direction to $\emptyset$.

After the notation has been presented, we are ready to demonstrate the algorithm. The algorithm consists of 6 rules. The node i in every rule is the local node.

1

If $n(i) \neq N(i)$
then $n(i) := N(i)$

2

If $\exists k \in N(i) \ : \ c(i, k) \neq N(i) \cap n(k)$
then $c(i, k) := N(i) \cap n(k)$

3a

If $\exists j \in N(i) \ : \ (|c(i, j)| = 1 \wedge (e(i, j, \emptyset) \neq -1 \vee opp(i, j, \emptyset) \neq k \vee opp(i, j, k) \neq \emptyset))$
then $e(i, j, \emptyset) := -1$
$opp(i, j, \emptyset) := k$
$opp(i, j, k) := \emptyset$
where $\{k\} = c(i, j)$

3b

If $\exists j \in N(i) \ : \ (|c(i, j)| = 1 \wedge (e(i, j, k) \neq d \vee v(i) \neq \max(|e(i, j, k)|, 1))$
then $e(i, j, k) := d$
$v(i) := \max(|e(i, j, k)|, 1)$
where

$$q = \begin{cases} -(1 + e(j, k, opp(j, k, i))) & \text{if } e(j, k, opp(j, k, i)) > 0, \\ e(j, k, opp(j, k, i)) & \text{otherwise} \end{cases}$$

$$d = \begin{cases} |e(i, k, opp(i, k, j))| & \text{if } |e(i, k, opp(i, k, j))| \geq |q|, \\ q & \text{otherwise} \end{cases}$$

$\{k\} = c(i, j)$

4

If $\exists j \in N(i)$: $(|c(i,j)| = 2 \wedge \exists k \in c(i,j)$: $(e(i,j,k) \neq d \vee opp(i,j,k) \neq l \vee opp(i,j,l) \neq k \vee v(i) \neq \max(|e(i,j,k)|, |e(i,j,l)|)))$
then $e(i,j,k) := d$
$opp(i,j,k) := l$
$opp(i,j,l) := k$
$v(i) := \max(|e(i,j,k)|, |e(i,j,l)|)$
where

$$q = \begin{cases} -(1 + e(j,k,opp(j,k,i))) & \text{if } e(j,k,opp(j,k,i)) > 0, \\ e(j,k,opp(j,k,i)) & \text{otherwise} \end{cases}$$

$$d = \begin{cases} |e(i,k,opp(i,k,j))| & \text{if } |e(i,k,opp(i,k,j))| \geq |q|, \\ q & \text{otherwise} \end{cases}$$

$\{k, l\} = c(i,j)$

5

If $\exists j, k \in N(i)$: $(k \in c(i,j) \wedge m(i,j,k) \neq MinEcc(i,j,k))$
then $m(i,j,k) := MinEcc(i,j,k)$

There is an assignment to the variable $n(i)$ in the rule 1 in order to allow each neighbor of the node i to know every other neighbor of i. Thanks to this, two adjacent nodes i and j can find out what their common neighbors are, and store this information in the variables $c(i,j)$ and $c(j,i)$ respectively, which is done in rule 2. Then the classical algorithm by Farley and Proskurowski can be adapted as rules 3a, 3b and 4. They calculate the eccentricities of nodes in a maximal outerplanar graph.

Once the eccentricities are calculated by rules 3a, 3b and 4, the rule 5 propagates the minimum eccentricity through all the graph.

Now the notion of a dual tree is used. The information about minimum eccentricity is propagated through the dual tree.

Note that in the rule 5 we used the function $MinEcc(i,j,k)$, which returns the correct value of $m(i,j,k)$. The $MinEcc$ function is defined as follows:

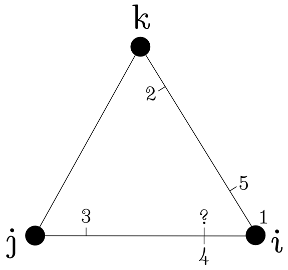

Figure 4: The visualization of computation of the function $MinEcc(i, j, k)$. The numbers stand for the order of checking (and assigning if necessary) values of $m(\cdot, \cdot, \cdot)$. The order above is: 1. $(v(i), \emptyset)$ (lines 1–2 of the $MinEcc$ function), 2. $m(k, i, j)$ (lines 3–5), 3. $m(j, i, k)$ (lines 6–8), 4. $m(i, j, opp(i, j, k))$ (lines 9–12), 5. $m(i, k, opp(i, k, j))$ (lines 13–16). The question mark stands for $m(i, j, k)$, which is the computed value.

Function $MinEcc(i,j,k)$

1 $v := v(i)$
2 $dir := \emptyset$
3 **if** $fst(m(k, i, j)) < v \wedge snd(m(k, i, j)) \in \{opp(k, j, i), \emptyset\}$ **then**
4 $\quad (v, dir) := m(k, i, j)$
5 **end if**
6 **if** $fst(m(j, i, k)) < v \wedge snd(m(j, i, k)) \in \{opp(j, k, i),\ \emptyset\ \}$ **then**
7 $\quad (v, dir) := m(j, i, k)$
8 **end if**
9 **if** $fst(m(i, j, opp(i, j, k))) < v \wedge snd(m(i, j, opp(i, j, k))) \neq k$ **then**
10 $\quad v := fst(m(i, j, opp(i, j, k)))$
11 $\quad dir := opp(i, j, k)$
12 **end if**
13 **if** $fst(m(i, k, opp(i, k, j)) < v \wedge snd(m(i, k, opp(i, k, j))) \neq j$ **then**
14 $\quad v := fst(m(i, k, opp(i, k, j))$
15 $\quad dir := opp(i, k, j)$
16 **end if**
17 **return** (v, dir)

Two projection functions are used in the above $MinEcc$ function: $fst((a, b)) \stackrel{\text{def}}{=} a$ and $snd((a, b)) \stackrel{\text{def}}{=} b$, which take the first and second element of the pair respectively.

The $MinEcc$ function computes the minimum value over eccentricities and the direction which it comes from for the triangle region specified by three parameters i, j, k (see Fig. 4). The first step is to consider the node i itself as a candidate with the minimum value of the eccentricity available in the neighborhood. In this case, the direction would be $\emptyset$ as if it did not come from a different region. In the second and third step, the node i checks as a candidate for the minimum value the values of $m(k, i, j)$ and $m(j, i, k)$ from the neighbor nodes k and j. If the values in the nodes k or j come from regions incident to the edges $\{i, k\}$ or $\{i, j\}$, they are not trusted. It is to ensure that no wrong value can last in the region for an infinitely long time (number of moves). And int the last two steps, the values from two neighbor regions (incident to i) are checked.

Lemma 7. *Algorithm consisting of rules 1–4 stabilizes in $\mathcal{O}(n^2)$ number of moves.*

Proof. The stabilization of rule 1 is obvious as the guard does not depend on the variables in the neighbor nodes. So rule 1 gets inactive in a finite time. Rule 2 depends only on static (after stabilizing rule 1) information computed by rule 1. Hence it stabilizes in a limited (i.e. constant) number of moves per node (as rule 1 does).

The same applies to rule 3a, as it depends on variable values computed by the two former rules, because it concerns an outerface edge (i.e. the edge belonging to the Hamilton cycle), which is the initial case of the recursive classical algorithm. Once the $c(i, j)$ is properly computed in the node i, it never changes. Thus, if any of the variables $e(i, j, \emptyset)$, $opp(i, j, \emptyset)$ or $opp(i, j, k)$ is in a wrong state, then all the variables are correctly computed and also never change.

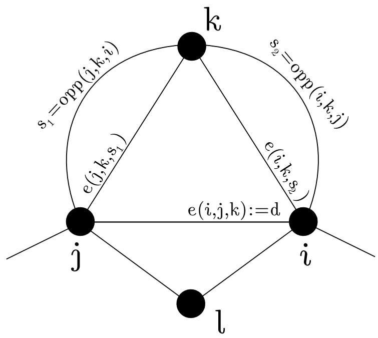

Figure 5: The visualization of rule 4.

Now all the nodes have got rules 3a and 3b inactive. Then we consider rule 4. Note that this rule is applicable only for graphs bigger than a triangle. Suppose there are two adjacent edges lying on an outerface of the graph. There has to be the third edge, which is also adjacent to them, and the edge eccentricities of this edge stabilize with rule 4. This is the first layer of correct rule 4 computation. Each subsequent layer of the correct computation of rule 4 depends on the previous layer. We have a finite graph, so rule 4 stabilizes. As each layer of computation of rule 4 takes $\mathcal{O}(n)$ moves and there are $\mathcal{O}(n)$ layers, it all takes $\mathcal{O}(n^2)$ moves. The last layer of computation stabilizes by rule 3b, as it reaches another outerface of the graph. □

Lemma 8. *If the phase 1–4 has stabilized a system, then phase 5 will stabilize in $\mathcal{O}(n^2)$ number of moves.*

Proof. The worst case would be if every dual vertex had a wrong value for the variable $v(i)$ (for example, as a result of starting up the system) and there were also wrong values of $m(\cdot,\cdot,\cdot)$ for every dual vertex (region of the graph). Let us assume that every $v(i)$ is less than the correct minimum eccentricity and there are no nodes i, j such that $v(i) = v(j)$. Consequently, the worst order of propagation of the $m(\cdot,\cdot,\cdot)$ values would be when the value $v(i)$ spreads first (for some i), being the biggest among all the other $v(k)$ (for all nodes k except for i), but still it is lower than the correct minimum eccentricity.

This propagation takes $\mathcal{O}(n)$ moves. Now the dual tree is filled with an incorrect value of some $v(i)$. But there are $n-1$ wrong candidates of minimum eccentricity left to spread. Once again, the pessimistic case would be when the next value to propagate was the maximal one among all the candidates, and lower than the one spread in the tree.

Each of these phases takes $\mathcal{O}(n)$ moves and there are $\mathcal{O}(n)$ phases, so it takes $\mathcal{O}(n^2)$ moves. □

The subsequent theorem results from lemmas 7 and 8, as well as from the fact that after each move made by the rules 1-4 the whole phase 5 can be run.

Theorem 9. *The algorithm takes $\mathcal{O}(n^4)$ number of moves to stabilize.*

Now we state the last property of our algorithm.

Theorem 10. *After stabilization of the algorithm the system is in a legitimate state.*

Proof. We shall use a proof by contradiction. Suppose that the system is in an illegitimate state. This means that the predicate of rule 5 is true. But this contradicts the fact that the system is stable. □

6 Conclusion

We have presented a distributed, self-stabilizing algorithm for locating the center of maximal outerplanar graphs. The algorithm requires $\mathcal{O}(n^4)$ moves to stabilize and stores the result of its computation in a distributed manner as a variable

$m(\cdot,\cdot,\cdot)$ (defined on p. 7) in each computing node. The value of the variable is minimum eccentricity and the direction pointing towards the center.

There is a question if the method used in [Turau 2012] would endow a self-stabilizing algorithm for locating the center of a maximal outerplanar graph with a lower complexity.

Acknowledgments

The authors would like to thank anonymous reviewers for constructive suggestions and comments. We are most grateful to one of the reviewers for referring us to the paper [Turau 2012].

References

[Bielak and Pańczyk 2012] Bielak, H., M. Pańczyk, M.: "A self-stabilizing algorithm for finding weighted centroid in trees"; Annales UMCS Informatica, AI XII, 2 (2012), 27–37.

[Blair and Manne 2003] Blair, J. R. S., Manne, F.: "Efficient self-stabilizing algorithms for tree networks"; Proceedings of the 23rd International Conference on Distributed Computing Systems, (2003), 20–27.

[Bruell et al. 1999] Bruell, S. C., Ghosh, S., Karaata, M. H., Pemmaraju, V.: "Self-stabilizing algorithms for finding centers and medians of trees"; SIAM Journal on Computing, 29 (1999), 600–614.

[Chepoi 2012] Chepoi, V., Fevat, T., Godard, E.,Vaxès, Y.: "A self-stabilizing algorithm for the median problem in partial rectangular grids and their relatives"; Algorithmica, 62 (2012), 146–168.

[Dijkstra 1974] Dijkstra, E. W.: "Self-stabilizing in spite of distributed control"; Communications of the ACM, 17 (1974), 643–644.

[Dolev 2000] Dolev, S.: "Self-stabilization"; The MIT Press (2000).

[Farley 1982] Farley, A. M.: "Vertex centers of trees"; Transportation Science, 16 (1982), 265–280.

[Farley and Proskurowski 1980] Farley, A. M., Proskurowski, A.: "Computation of the center and diameter of outerplanar graphs"; Discrete Applied Mathematics, 2 (1980), 185–191.

[Fleischner et al. 1974] Fleischner, H. J. , Geller, D. P., Harary, F.: "Outerplanar graphs and weak duals"; Journal of the Indian Mathematical Society, 38 (1974), 215–219.

[Goldman 1972] Goldman, A. J.: "Minimax location of a facility in a network"; Transportation Science, 6 (1972), 407–418.

[Harary 1972] Harary, F.: "Graph Theory"; Addison-Wesley, 1972.

[Hedetniemi et al. 1981] Hedetniemi, S. M., Cockayne, E. J., Hedetniemi, S. T.: "Linear algorithms for finding the jordan center and path center of a tree"; Transportation Science, 15 (1981), 98–114.

[Kariv et al. 1979a] Kariv, O., Hakimi, S. L.: "An algorithmic approach to network location problems. I: The p-centers"; SIAM J. Applied Mathematics, 37 (1979), 513–538.

[Kariv et al. 1979b] Kariv, O., Hakimi, S. L.: "An algorithmic approach to network location problems. II: The p-medians"; SIAM J. Applied Mathematics, 37 (1979), 539–560.

[Korach et al. 1984] Korach, E., Rotem, D., Santoro, N.: "Distributed algorithms for finding centers and medians in networks"; ACM Transactions on Programming Languages and Systems, 6 (1984), 380–401.

[Laskar et al. 1983] Laskar, R., Shier, D.: "On powers and centers of chordal graphs"; Discrete Applied Mathematics, 6 (1983), 139–147.

[Proskurowski 1980] Proskurowski, A.: "Centers of maximal outerplanar graphs"; Journal of Graph Theory, 4 (1980), 75–79.

[Rosenthal et al. 1989] Rosenthal, A., Pino, J.: "A generalized algorithm for centrality problems on trees"; JACM, 36 (1989), 349–361.

[Schneider 1993] Schneider, M.: "Self-stabilization"; ACM Computing Surveys, 25, 1, (1993).

[Turau 2012] Turau, V.: "Efficient transformation of distance-2 self-stabilizing algorithms"; Journal of Parallel and Distributed Computing, 72 (2012), 603–612.