1998 International Computer Symposium
Workshop on Algorithms
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

AN APPLICATION OF KARAATA'S SELF-STABILIZING CENTER-FINDING ALGORITHM

Tetz C. Huang, Ji-Cherng Lin and Hsueh-Jen Chen

Department of Computer Science & Information Engineering,
Yuan-Ze University, Chung-Li, Taiwan, R.O.C.
Email:{cstetz, csjclin}@cs.yzu.edu.tw, s871951@mail86.yzu.edu.tw

ABSTRACT

In this paper, we design a *self-stabilizing* algorithm which finds a *2-center* for a distributed system with a tree topology. Our algorithm is based on the algorithm in [14,15]. The latter algorithm enables us to find the center (or centers) for the tree. If we sever the tree at the center (or centers), we obtain two subtrees. One of the major works in this paper is to show that if we pick a center from each subtree, the two picked centers will constitute a 2-center for the original tree. With this in mind, we design our algorithm, so that it is equipped with the ability of " sensing " the two subtrees and then finding out a center in each of them.

Keywords: Distributed systems, self-stabilizing algorithms, centers, 2-centers, trees

1. INTRODUCTION

The notion of self-stabilization in distributed system first appeared in the classic paper by E.W. Dijkstra [3]. According to him, a distributed system is self-stabilizing if regardless of any initial state, the system can adjust itself automatically to eventually reach a legitimate state in a finite number of steps and then remain so thereafter until it is incurred a subsequent transient failure.

In this paper, we propose a self-stabilizing distributed system with a tree topology. The goal of our system is to identify a 2-center for the tree system. Therefore, we define the legitimate state to be those states in which a 2-center for the system can be identified. The algorithm in our system utilizes heavily the center-finding algorithm in Karaata et al. [15]. As in [15], our system also assumes the existence of a central demon.

The rest of this paper is arranged as follows. In Section 2, some relevant information about Karaata's algorithm is presented. In Section 3.1, the theoretical foundation of our system is established. In Section 3.2, the algorithm of our system is proposed. Section 3.3 explains our algorithm. Finally in Section 3.4, some words about correctness conclude this exposition.

2. KARAATA'S ALGORITHM

Since our algorithm is based on Karaata's algorithm which finds the center (or centers) of a tree, some useful information about Karaata's algorithm will be presented here for later reference.

We note first that the underlying topology of Karaata's system is a tree; the vertices of the tree represent processors; the system assumes the existence of a central demon who can randomly select one among all the privileged processors to make a move; the central demon need not be fair in any sense.

Next, we define the concept of a center of a tree. Let $T = (V, E)$ be a tree. For $i, j \in V$, let $d(i, j)$ denote the distance between i and j, i.e., the length of the unique simple path in T which connects i and j. Let $e(i) = max\ \{d(i,j) \mid j \in V\}$ denote the *eccentricity* of a vertex i, i.e., the distance between i and a farthest vertex from i in T. A center of T is a vertex with the minimum eccentricity.

The following proposition states a well-known property regarding centers of trees. The proof of this can be found in theorem 2.1 of [1].

Proposition 1 A tree has a single center or two adjacent centers (cf. Figure 1 and Figure 2).

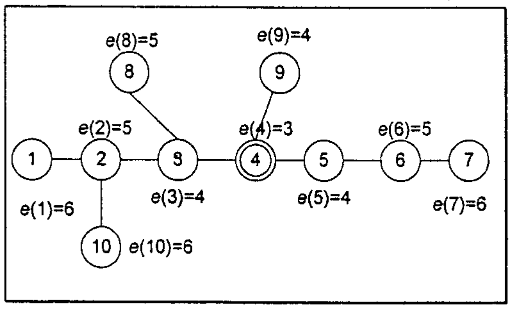


Figure 1 Eccentricities of vertices in a tree. The only center of the tree is marked by double circles.

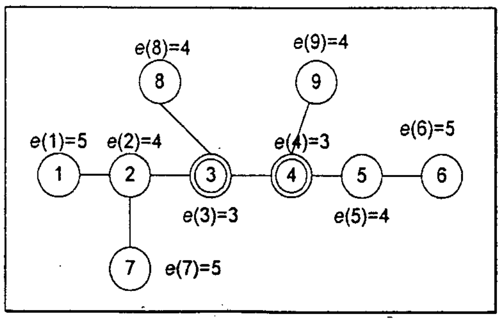


Figure 2 Eccentricities of vertices in a tree. Here, the tree has two centers, which are marked by double circles.

In order to describe the algorithm, we first define some notations: $h(i)$ is a local variable of the vertex i, called the h-value of vertex i. $N(i) = \{ j \in V \mid \{i, j\} \in E \}$ denotes the set of neighbors of vertex i. $N_h(i) = \{ h(j) \mid j \in N(i) \}$ denotes the *multi-set* of h-values of the neighbors of i. $N_h^-(i) = N_h(i) - \{ max(N_h(i)) \}$ denotes all of $N_h(i)$ with one maximum h-value removed. For example, if $N_h(i) = \{3,4,4\}$, then $N_h^-(i) = \{3,4\}$.

To facilitate the description of this algorithm, we introduce the following condition on the h-value of vertex i, called the height condition: we say that vertex i satisfies the height condition if

(1) i is a leaf and $h(i) = 0$ or

(2) i is not a leaf and $h(i) = 1 + max\,N_h^-(i)$

The following is the Karaata's algorithm:

```
[ ( $i$ is a leaf) $\wedge h(i) \neq 0 \rightarrow h(i) := 0$
  ( $i$ is not a leaf ) $\wedge h(i) \neq 1 + max\,N_h^-(i)$
    $\rightarrow h(i) := 1 + max\,N_h^-(i)$ ]
```

When all the vertices in T satisfy the height condition, we say that the system is in legitimate state, and the center (or centers) of T is the only vertex whose h-value is greater than or equal to the h-values of all neighboring vertices (cf. Figure 3 and Figure 4). It is clear that the purpose of Karaata's algorithm is to ensure that each vertex satisfies the height condition.

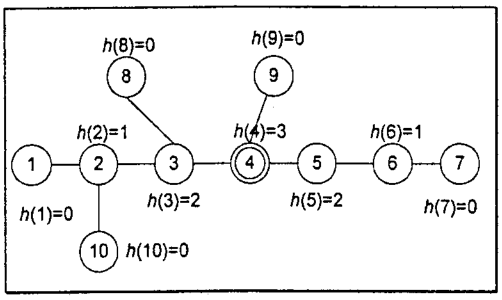


Figure 3 A legitimate state with only one center. Every vertex in the tree satisfies the height- condition.

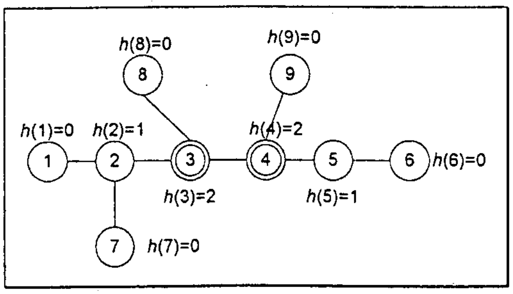


Figure 4 A legitimate state with two centers. Every vertex in the tree satisfies the height-condition.

In [15], the convergence property of the algorithm is proved. The closure property of the algorithm is trivial, because one can easily see that when the system reaches a legitimate state, every processor in the system satisfies its height condition and will not be privileged any more. Thus, the whole system is in deadlock and will stay in the legitimate state.

3. OUR ALGORITHM

As mentioned previously, our goal is to design a self-stabilizing distributed system with a tree topology which can identify a 2-center of itself. So, the underlying topology of our system is a tree; the vertices represent processors; our system also assumes the existence of a central demon who can select one among all the privileged processors to make a move; but, unlike Karaata's system, our central demon is fair, i.e., under such a central demon, there will not exist any infinite sequence of moves in our system in which certain processor only moves finitely many times.

3.1 Theoretical Foundation

We begin this subsection with some notations and terminology. Let $T = (V, E)$ be a tree. A subset of V which consists of 2 vertices will be called a 2-set in T. If X is a 2-set in T and v is a vertex in T, then the distance between X and v, $d(X,v) = \min_{x \in v} d(x,v)$ and eccentricity of X, $e(X) = \max_{v \in V} d(X,v)$. A 2-center of T is a 2-set in T with the minimum eccentricity. For example, let $X=\{2, 5\}$, Figure 5 shows the distance between X and all nodes in T, and we know the eccentricity of X, $e(X)=2$. In Figure 6, $X_r = \{3, 5\}$ is a 2-center of T because it has the minimum eccentricity among all the 2-sets in T.

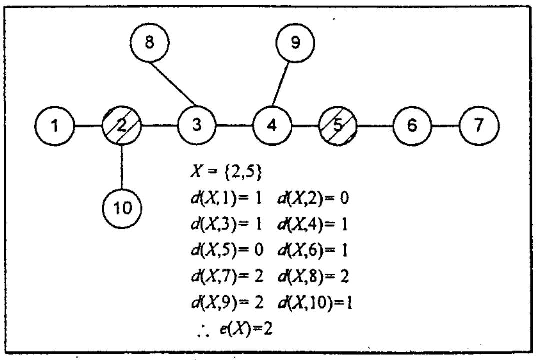


Figure 5 The eccentricity of a 2-set X.

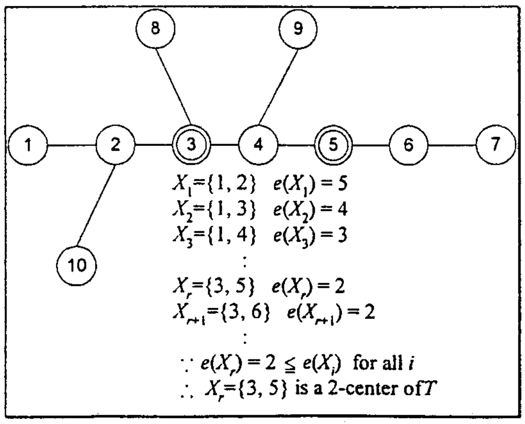


Figure 6 A 2-center of T.

First we need a lemma about centers of trees. One can easily see that this lemma is a stronger version of previous proposition 1.

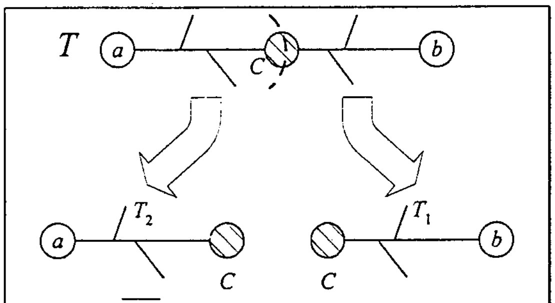
Lemma 1 Let L be a longest simple path of T.
Then (1) if the length of L is $2l$, then T has a unique center C, namely, the midpoint of L, and $e(C) = l$;
(2) if the length of L is $2l+1$, then T has 2 centers C_1 and C_2, namely, the two midpoints of L, and $e(C_1)=e(C_2)=l+1$.

According to the number of centers and the degree of the center (or centers), we have the following five cases to consider:

(1) when T has one center C and $deg(C) = 2$
(2) when T has one center C and $deg(C) \geqq 3$
(3) when T has two centers C_1, C_2 and $deg(C_1)= deg(C_2)= 2$
(4) when T has two centers C_1, C_2 and one of the centers has the degree greater than or equal to 3
(5) when T has two centers C_1, C_2 and $deg(C_1) \geqq 3$, $deg(C_2) \geqq 3$

In each of the above five cases, we partition the tree T into two subtrees, T_1 and T_2.

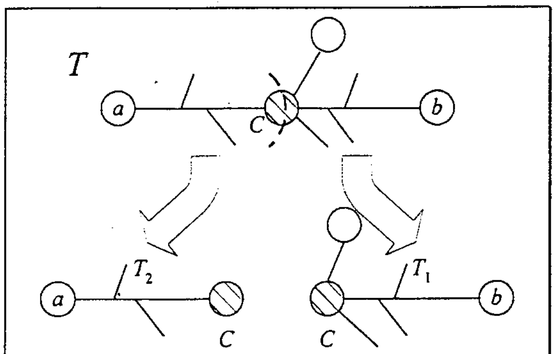
(1) When T has one center C and $deg(C)=2$



The path $\overline{aCb}$ represents a longest simple path of T.

Figure 7

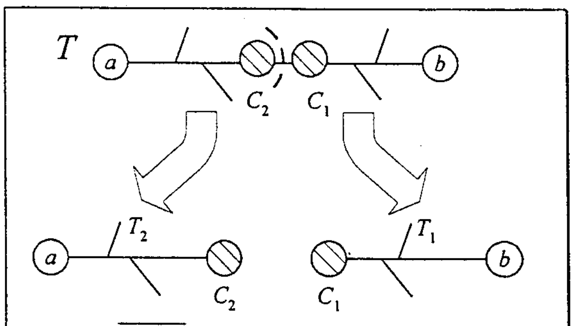
(2) When T has one center C and $deg(C) \geqq 3$



The path $\overline{aCb}$ represents a longest simple path of T.

Figure 8

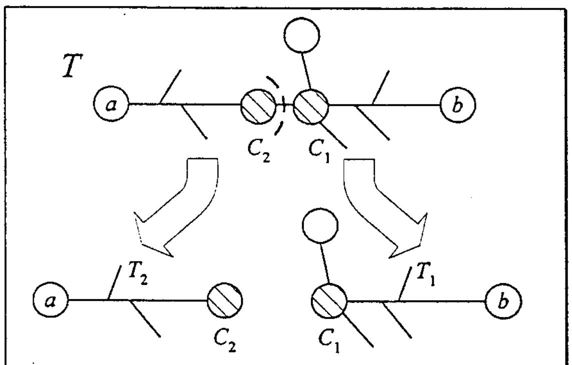
(3) When T has two centers C_1, C_2 and $deg(C_1)=deg(C_2)= 2$



The path $\overline{aC_2C_1b}$ represents a longest simple path of T.

Figure 9

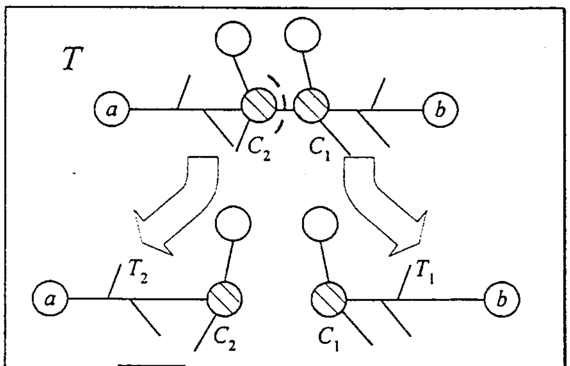
(4) When T has two centers C_1, C_2 and one of the centers has the degree greater than or equal to 3



The path $\overline{aC_2C_1b}$ represents a longest simple path of T.

Figure 10

(5) When T has two centers C_1, C_2 and $deg(C_1) \geqq 3$, $deg(C_2) \geqq 3$



The path $\overline{aC_2C_1b}$ represents a longest simple path of T.

Figure 11

The main theorem of this paper claims that in all the above five cases, if B_1 is a center of T_1 and B_2 is a center of T_2, then $\{B_1, B_2\}$ is a 2-center of the original tree T.

There are 17 cases to prove:

1. T has only one center C, $deg(C)$=2, and T_1 as well as T_2 has only one center.
2. T has only one center C, $deg(C)$=2, and one of T_1 and T_2 has two centers, while the other has only one center.
3. T has only one center C, $deg(C)$=2, and T_1 as well as T_2 has two centers.
4. T has only one center C, $deg(C)\geqq 3$, and T_1 as well as T_2 has only one center.
5. T has only one center C, $deg(C)\geqq 3$, and T_1 has only one center, while T_2 has two centers.
6. T has only one center C, $deg(C)\geqq 3$, and T_1 has two centers, while T_2 has only one center.
7. T has only one center C, $deg(C)\geqq 3$, and T_1 as well as T_2 has two centers.
8. T has two centers C_1, C_2, $deg(C_1)$= $deg(C_2)$= 2, and T_1 as well as T_2 has only one center
9. T has two centers C_1, C_2, $deg(C_1)$= $deg(C_2)$= 2, and one of T_1 and T_2 has two centers, while the other has only one center.
10. T has two centers C_1, C_2, $deg(C_1)$= $deg(C_2)$= 2, and T_1 as well as T_2 has two centers.
11. T has two centers C_1, C_2, one of C_1 and C_2 with the degree is greater than or equal to 3, and T_1 as well as T_2 has only one center.
12. T has two centers C_1, C_2, one of C_1 and C_2 with the degree is greater than or equal to 3, and T_1 has only one center, while T_2 has two centers.
13. T has two centers C_1, C_2, one of C_1 and C_2 with the degree is greater than or equal to 3, and T_1 has two centers, while T_2 has only one center.
14. T has two centers C_1, C_2, one of C_1 and C_2 with the degree is greater than or equal to 3, and T_1 as well as T_2 has two centers.
15. T has two centers C_1, C_2, $deg(C_1)\geqq 3$, $deg(C_2)\geqq 3$, and T_1 as well as T_2 has only one center.
16. T has two centers C_1, C_2, $deg(C_1)\geqq 3$, $deg(C_2)\geqq 3$, and one of T_1 and T_2 has two centers, while the other has only one center.
17. T has two centers C_1, C_2, $deg(C_1)\geqq 3$, $deg(C_2)\geqq 3$, and T_1 as well as T_2 has two centers.

Based on the above theoretical foundation, our algorithm is therefore designed to search for the center (or centers) of T first and then search for the centers B_1 and B_2 of the subtrees T_1 and T_2 respectively, because we then get a 2-center $\{B_1, B_2\}$ of T.

3.2 The Algorithm

In order to describe our algorithm, we introduce three local variables of each processor i as follows: $h_1(i)$ and $h_2(i)$ are local variables of vertex i with nonnegative integer values, and $a(i)$ is a local variable of vertex i, which indicates the identity of one of i's neighbors or is set to be ∞, depending on situations. In addition, we will use the following notations. We define $N_{h_2}(i)$ as follows:

$$N_{h_2}(i)=\begin{cases}\{h_2(j)\mid j\in N(i)\wedge j\neq a(i)\} & \text{if } (h_1(i)\geq \max N_{h_1}(i) \\ & \wedge \deg(i)\geq 3) \\ (\{h_2(j)\mid j\in N(i)\}-\{h_2(k)\})\cup\{0\} & \text{if } (h_1(i)<\max N_{h_1}(i)) \\ & \wedge \exists! k\in N(i) \text{ s.t. } i=a(k) \\ \{h_2(j)\mid j\in N(i)\} & \text{otherwise}\end{cases}$$

Definitions of $N_{h_1}(i)$ and $N^-_{h_1}(i)$ and $N^-_{h_2}(i)$ are analogous to those of $N_h(i)$ and $N^-_h(i)$ in Karaata's algorithm in section 2. Height-conditions on h_1-value and height-condition on h_2-value are defined analogously to the height-condition on h-value in section 2.

The following four predicates will be used in our algorithm to set priorities.

$P1 \equiv deg(i) = 1 \wedge h_1(i) = 0$

$P2 \equiv deg(i) \geqq 2 \wedge h_1(i) = 1 + max\,N^-_{h_1}(i)$

$P3 \equiv deg(i) \geqq 3 \wedge h_1(i) \geqq max\,N_{h_1}(i) \wedge a(i) =$
$max\{j \in N(i)|h_1(j) = max\,N_{h_1}(i)\}$

$P4 \equiv [h_1(i) < max\,N_{h_1}(i) \vee (h_1(i) \geqq max\,N_{h_1}(i)$
$\wedge deg(i) = 2)] \wedge a(i) = \infty$

Now, our algorithm is ready.

$R1$: $deg(i) = 1 \wedge h_1(i) \neq 0 \rightarrow h_1(i) := 0$

$R2$: $deg(i) \geqq 2 \wedge h_1(i) \neq 1 + max\,N^-_{h_1}(i)$
$\rightarrow h_1(i) := 1 + max\,N^-_{h_1}(i)$

$R3$: $(P1 \vee P2) \wedge deg(i) \geqq 3 \wedge h_1(i) \geqq max\,N_{h_1}(i)$
$\wedge a(i) \neq max\{j \in N(i)|h_1(j) = max\,N_{h_1}(i)\}$
$\rightarrow a(i) := max\{j \in N(i)|h_1(j) = max\,N_{h_1}(i)\}$

$R4$: $(P1 \vee P2) \wedge [h_1(i) < max\,N_{h_1}(i)$
$\vee (h_1(i) \geqq max\,N_{h_1}(i) \wedge deg(i) = 2)] \wedge a(i) \neq \infty$
$\rightarrow a(i) := \infty$

$R5$: $(P1 \vee P2) \wedge (P3 \vee P4) \wedge [deg(i) = 1$
$\vee (h_1(i) \geqq max\,N_{h_1}(i) \wedge deg(i) = 2)] \wedge h_2(i) \neq 0$
$\rightarrow h_2(i) := 0$

$R6$: $(P1 \vee P2) \wedge (P3 \vee P4) \wedge [(h_1(i) \geqq max\,N_{h_1}(i)$
$\wedge deg(i) \geqq 3) \vee (deg(i) \geqq 2 \wedge h_1(i) < max\,N_{h_1}(i))]$
$\wedge h_2(i) \neq 1 + max\,N^-_{h_2}(i)$
$\rightarrow h_2(i) := 1 + max\,N^-_{h_2}(i)$

3.3 Description and Explanation

Each processor in the system is equipped with all the above 6 rules. Note first that $R1$ and $R2$ are exactly the Karaata's algorithm on h_1-value, the goal of which is to identify the center (or centers) of the tree T. If there is no obstacle which prevents the system from continually applying $R1$ and $R2$ to processors, the above goal will be eventually accomplished, i.e., the system will eventually, in a finite number of steps, reach a state in which all processors of the system satisfy the height-conditions on h_1-value. When this happen, we shall say that the system has reached GOAL ONE. Then, the center (or centers) can be identified because it is the only processor (or processors) in the system which has its h_1-value greater than or equal to those of its neighbors; and thereafter, rules $R1$ and $R2$ will not be enabled any more and h_1-values of all processors will stay static. Since in each processor, due to the device of our algorithm, the rules $R1$ and $R2$ have the first priority of being enabled, and since the central demon in our system is a fair one, rules $R1$ and $R2$ can be continually applied to processors in the system until they eventually cause the system to reach GOAL ONE, in a finite number of steps. We should reiterate here that the role of the fair demon is to prevent the system, when it has not reached GOAL ONE yet, from continuously executing rules other than $R1$ and $R2$, while not executing $R1$ and $R2$ at all, and thus producing an obstacle for the system to reach GOAL ONE.

From the above discussion, we see that the priority of $R1$ and $R2$ together with the fair demon ensure that the system will reach GOAL ONE in a finite number of steps. So from now on, we may assume that the h_1-values of all the processors in the systems satisfy the height-conditions and stay static. The center (or centers) can then be identified by examining h_1-values and the tree T can be suitably divided into two subtrees T_1 and T_2. We then apply Karaata's algorithm, using h_2-values, on both subtrees, in order to find centers in them. $R5$ and $R6$ serve for this purpose while $R3$ and $R4$ serve for solving some technical difficulties which will be discussed later.

There are five cases (as mentioned previously) to discuss. Here we explain in details for case 1 only. For other cases, the explanations are similar.

Case 1 (when T has only one center C and $deg(C) = 2$ (cf. Figure7))

First, T is divided at the center C into two subtrees T_1 and T_2, in an obvious way. The key point in this case is that the center C should be treated as a leaf node in both subtrees T_1 and T_2. For this reason, when we apply Karaata's algorithm, using h_2-values, on T_1 and T_2, in order to find centers in T_1 and T_2, there will be no conflict between dealing with C in T_1 and dealing with C in T_2, because on both occasions, we need to do the same thing—setting the h_2-value of C to be zero. $R4$, $R5$ and $R6$, but not $R3$, apply in this case. One can easily see that the system will converge, in a finite number of steps, to a legitimate state, i.e., (1) all processors will satisfy the height-conditions on h_1-value and h_2-value except that the center C has its own h_2-value equal to zero, and (2) the a-values of all nodes will become ∞. Then no processor in the system will be privileged any more and the whole system is in deadlock and stay in the legitimate state. At this time, centers in both subtrees T_1 and T_2 can be identified, because they are the only nodes in T whose h_2-values are greater than or equal to the h_2-values of their neighboring vertices. Once centers in both subtrees are identified, a 2-center of the original tree T is also identified, according to our main theorem.

3.4 Correctness

To show our algorithm is self-stabilizing, we need to show (1) the convergence property, i.e., regardless of the initial state and regardless of the privilege selected each time for the next move, our system is guaranteed to find itself in a legitimate state after a finite number of moves, and (2) the closure property, i.e., once the system arrives at a legitimate state, it will stay in legitimate states unless a subsequent transient error occurs. However, since all this can be seen easily from the description and explanation in subsection 3.3, there is really nothing more to say.

References

[1] F. Buckley and F. Harary. *Distance in Graphs*, Addison-Wesley Publishing Company, Redwood City, CA, 1990.

[2] N. Christofides. *GRAPH THEORY: An Algorithmic Approach*, pages 101.

[3] EW Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, Vol. 17, pages 643-644, 1974.

[4] EW Dijkstra. EWD391 Self-stabilization in spite of distributed control. *Selected Writings on Computing: A Personal Perspective*, pages 41-46, 1982. EWD391's original date is 1973.

[5] EW Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, Vol. 1, pages 5-6, 1986.

[6] S Dolev, A Israeli, and S Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, Vol. 7, pages 3-16, 1993.

[7] AK Datta, TF Gonzalez, and V Thiagarajan. Self-stabilizing algorithms for tree metrics. *ICAPP95 IEEE First International Conference on Algorithms and Architectures for Parallel Processing*, pages 471-479, 1995.

[8] AM. Farley. Vertex centers of trees. *Transportation Science*, Vol. 16, pages 265-280, 1982.

[9] G.Y.Handler. Medi-Centers of a Tree. *Transportation Science*, Vol. 19, No. 3, pages 246-260, Aug. 1985.

[10] S.C. Hsu, S.T. Huang. A generalized self-stabilizing protocol for centrality problem on tree networks. *Proceedings of National Computer Symposium*, pages 59-64 1991.

[11] ANC Kang and D.A Ault. Some properties of a Centroid of A Free Tree. *Information Processing Letters*, Vol. 4, No. 1, pages 18-20, Sep. 1975.

[12] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Transactions on Programming Languages and Systems*, Vol. 6, pages 380-401, 1984.

[13] JLW Kessels. An exercise in proving self-stabilization with a variant function. *Information Processing Letters*, Vol. 29, pages 39-42, 1988.

[14] MH Karaata, SV Pemmaraju, SC Bruell, and S Ghosh. Self-stabilizing algorithms for finding centers and medians of trees. *PODC94 Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing,* pages 374, 1994.

[15] MH Karaata, SV Pemmaraju, SC Bruell , and S Ghosh. Self-stabilizing algorithms for finding centers and medians of trees. *Technical Report, TR94-03*, University of Iowa, 1994.

[16] M Schneider. Self-stabilization. *ACM Computing Surveys*, Vol. 25, pages 45-67, 1993.

[17] LC Wuu and ST Huang. Distributed self-stabilizing systems. *Journal of Information Science and Engineering*, Vol. 11, pages 307-319, 1995.