



ELSEVIER

Information Processing Letters 80 (2001) 221–223

Information
Processing
Letters

www.elsevier.com/locate/ipl

Maximal matching stabilizes in time $O(m)$

Stephen T. Hedetniemi, David P. Jacobs*, Pradip K. Srimani

Department of Computer Science, Clemson University, Clemson, SC 29634-0974, USA

Received 11 December 2000; received in revised form 14 February 2001

Communicated by A.A. Bertossi

Abstract

On a network having m edges and n nodes, Hsu and Huang's self-stabilizing algorithm for maximal matching stabilizes in at most $2m + n$ moves. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Analysis of algorithms; Matching; Self-stabilizing

1. Introduction

In [1] Hsu and Huang present a self-stabilizing algorithm for constructing a maximal matching in a distributed network and show that it stabilizes in $O(n^3)$ time. In [2] Tel shows that, in fact, the Hsu–Huang algorithm runs in $O(n^2)$ time. More precisely, Tel shows that for any network having n nodes, the Hsu–Huang algorithm will stabilize within $\frac{1}{2}n^2 + 2n + 1$ moves. A family of graphs and accompanying executions are given that take $\frac{1}{2}n^2 + n - 2$ moves, showing that the bound is almost tight.

A network can be modeled with an undirected graph $G = (V, E)$, where V is its node set and E is its edge set. In this note we show that for any network having m edges and n nodes, the Hsu–Huang algorithm must stabilize in at most $2m + n$ moves. For sparse networks such as trees, our result implies an $O(n)$ bound. In both papers [1] and [2] a variant function is used for the complexity analysis. Our analysis uses a new proof

technique that counts the number of moves that occur on a given edge.

2. Proof of the bound

In self-stabilizing algorithms, each node maintains variables which determine its *local state*. The system's *global state* is the union of all local states. A node may change its local state by making a *move*. We follow the serial model of [1], so that no two nodes move at the same time. An execution will be represented as a sequence of moves m_1, m_2, \dots , in which m_s denotes the s th move. At any time, the system's global state is determined by its initial state and this sequence of moves.

If i is a node, then $N(i)$, its *open neighborhood*, denotes the set of nodes to which i is adjacent. Every node $j \in N(i)$ is called a *neighbor* of node i . In the Hsu–Huang algorithm, each node i maintains a single variable which is either null, denoted $i \rightarrow \text{null}$, or points to one of its neighbors j , denoted $i \rightarrow j$. The algorithm at node i is given by the following three rules.

* Corresponding author.

E-mail addresses: hedet@cs.clemson.edu (S.T. Hedetniemi), dpj@cs.clemson.edu (D.P. Jacobs), srimani@cs.clemson.edu (P.K. Srimani).

Algorithm 2.1. HSUHUANG()

R1: if $(i \rightarrow \text{null}) \wedge (\exists j \in N(i))(j \rightarrow i)$
 then set $(i \rightarrow j)$

R2: if $(i \rightarrow \text{null}) \wedge (\forall k \in N(i))(\neg(k \rightarrow i)) \wedge$
 $(\exists j \in N(i))(j \rightarrow \text{null})$
 then set $(i \rightarrow j)$

R3: if $(i \rightarrow j) \wedge (j \rightarrow k) \wedge (k \neq i)$
 then set $(i \rightarrow \text{null})$

In each of the three types of moves that node i might make, there is a *forcing neighbor* j which allows i to move. We use (i, j, R_k) to denote the execution of rule R_k , $1 \leq k \leq 3$, by node i , where j is its forcing neighbor. Such a move will be called an ij -move. Let m_1, m_2, \dots, m_q be an entire execution of moves. If $e = \{i, j\}$ is an (undirected) edge, let $c(i, j)$ denote the number of indices s for which m_s is an ij -move. Our analysis counts the moves that occur on each edge, so we define $c(e) = c(i, j) + c(j, i)$.

After the execution of (i, j, R_1) or (j, i, R_1) , we will say i and j are *matched*. The following is straightforward.

Lemma 1. *Once nodes i and j are matched, neither i nor j can move again.*

Lemma 2. *After (i, j, R_2) , at most one more ij -move, namely (i, j, R_3) , can occur.*

Proof. Let $m_s = (i, j, R_2)$, and suppose that there is at least one more ij -move. Let $m_t = (i, j, R_k)$ be the next ij -move. Clearly $m_t = (i, j, R_3)$. It suffices to show that no ij -move can occur after move t . After m_s , we must have

$$i \rightarrow j \wedge j \rightarrow \text{null}.$$

Just prior to move m_t we must also have

$$i \rightarrow j \wedge j \rightarrow k$$

for some $k \neq i$. Thus, there must exist some $r, s < r < t$, where either $m_r = (j, k, R_1)$ or $m_r = (j, k, R_2)$. But $m_r \neq (j, k, R_2)$, because just prior to move r we have $i \rightarrow j$. Therefore, $m_r = (j, k, R_1)$, implying that after move r , nodes j and k are matched. By Lemma 1, this precludes any subsequent ij -moves after move t . \square

Lemma 3. *Following (i, j, R_2) , there is exactly one more move on the same edge, namely either (j, i, R_1) or (i, j, R_3) .*

Proof. There must be at least one more ij - or ji -move since (i, j, R_2) leaves the system unstable. If the next move is a ji -move, it must be (j, i, R_1) , and by Lemma 1 there are no further moves on this edge. On the other hand, if the next move is an ij -move, it must be (i, j, R_3) , and by Lemma 2 there are no further ij -moves. As the proof of Lemma 2 shows, when (i, j, R_3) occurs, node j is matched with some node $k \neq i$, so no future ji -moves will occur either. \square

Lemma 4. *Following a move (i, j, R_3) , there are at most two more moves on edge $e = \{i, j\}$.*

Proof. If there is another move on edge $\{i, j\}$, clearly node j 's variable must first become null by executing (j, k, R_3) . After this, since both i and j will have null pointers, any following move could only be (i, j, R_2) or (j, i, R_2) . By Lemma 3 there will be exactly one more move. \square

Consider the initial state of each node i , and define $e = \{i, j\}$ to be an *initial edge* if $i \rightarrow j$. Letting I be the set of all initial edges, note that $|I| \leq n$.

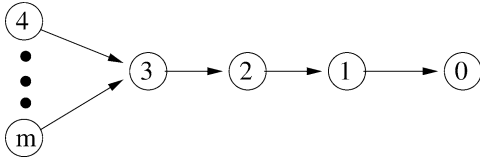
Lemma 5. *For each edge e , $c(e) \leq 3$, and for at most n edges, $c(e) = 3$.*

Proof. If $c(e) > 0$ then there is a first move m . Depending on whether $m = (i, j, R_1)$, $m = (i, j, R_2)$, or $m = (i, j, R_3)$, Lemmas 1, 3, and 4 show that $c(e)$ is at most three. To prove the second assertion, let A be the set of edges $\{e \mid c(e) = 3\}$. If $e \in A$, then the first move on e is of the form (i, j, R_3) . This implies that the initial state of node i is $i \rightarrow j$, and so $e \in I$. Therefore $A \subseteq I$, and so $|A| \leq n$. \square

Theorem 1. *For any network having m edges and n nodes, the Hsu–Huang algorithm stabilizes within $2m + n$ moves.*

Proof. This follows directly from Lemma 5. \square

It is worth noting that if $I = \emptyset$, then stabilization must occur in at most $2m$ moves.

Fig. 1. Tree T_m .

Corollary 1. *For any class of graphs having $O(n)$ edges, the Hsu–Huang algorithm stabilizes in $O(n)$ time.*

This corollary implies that the Hsu–Huang algorithm runs in linear time for many important classes of graphs, including graphs having nodes of bounded degree, and planar graphs.

3. Tightness of the bound

We now exhibit an infinite family of trees for which the Hsu–Huang algorithm can take $2m + n - 5$ moves, thus showing that the bound of the previous section is almost tight. Consider the tree T_m of Fig. 1 having

m edges, with initial state as shown by the arrows. All nodes except node 0 are pointing to a neighbor. Consider the following execution.

$(4, 3, R_3), (5, 3, R_3), \dots$

$(m, 3, R_3), (3, 2, R_3), (2, 1, R_3)$

$(4, 3, R_2), (5, 3, R_2), \dots$

$(m, 3, R_2), (2, 3, R_2), (3, 2, R_1)$

$(4, 3, R_3), (5, 3, R_3), \dots$

$(m, 3, R_3), (0, 1, R_1)$

Among the m edges, there are two edges $e = \{2, 1\}$ and $e = \{1, 0\}$ with $c(e) = 1$, and $m - 2$ edges with three moves. Thus the execution takes $2m + n - 5$ moves.

References

- [1] S.-C. Hsu, S.-T. Huang, A self-stabilizing algorithm for maximal matching, Inform. Process. Lett. 43 (1992) 77–81.
- [2] G. Tel, Maximal matching stabilizes in quadratic time, Inform. Process. Lett. 49 (1994) 271–272.