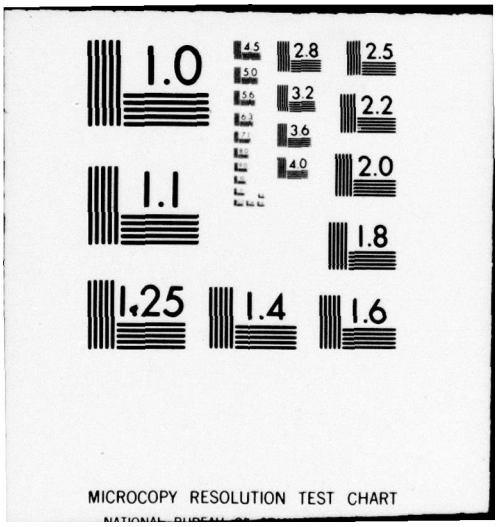
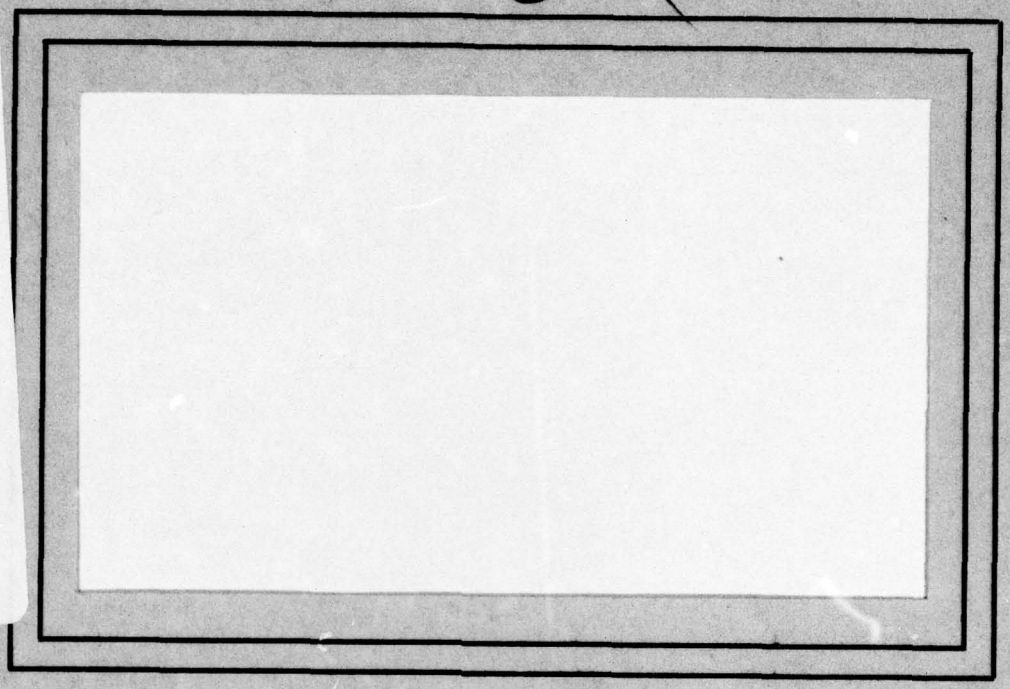
AD-A077 389 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 9/2
CELLULAR D-GRAPH AUTOMATA WITH AUGMENTED MEMORY.(U)
JUN 79 T DUBITZKI , A WU AFOSR-77-3271
UNCLASSIFIED TR-771 AFOSR-TR-79-1164 NL

1 OF 1
ADA 077389

END
DATE FILMED
12-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART

LEVEL II



UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

TR-771
AFOSR-77-3271

June 1979

CELLULAR D-GRAPH AUTOMATA WITH AUGMENTED MEMORY

Tsvi Dubitzki
Angela Wu
Computer Science Center
University of Maryland
College Park, MD 20742

See 1473 in back.

ABSTRACT

This paper deals with cellular d-graph automata in which each node has internal memory proportional to $\log_{d-1} N$ where N is the number of nodes in the underlying graph G. It is shown how such an automaton can assign unique labels to the nodes of G in $O(\log_{d-1} N)$ time. Such an automaton can also count the number of nodes and edges in G in $O(\log_{d-1} N)$ time. Algorithms for identifying all the cut nodes and all bridges in G, each in $O(\log_{d-1} N)$ time, are also presented.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper.

1. Introduction

Informally, a cellular d-graph automaton [1] is a graph of bounded degree with a finite-state automaton at each node. Many results on the graph recognition capabilities of such automata were established in [1]. This paper investigates an extended class of these automata in which the processor at each node has an amount of memory which is allowed to grow with the logarithm of the number of nodes.

We first review some of the basic concepts introduced in [1]. A <u>d-graph</u> is a graph with bounded degree d. A <u>cellular d-graph automaton</u> $\bar{M}$ is a triple (G,M,H) where G is a d-graph (P,A,f,g) defined on the label set $\Lambda=(1,2,\ldots)$.

P is the set of nodes in G.

$A \subseteq P \times P$ is a symmetric relation on P called the set of edges

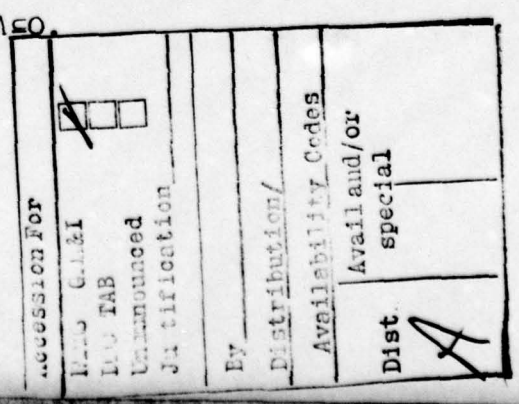
$f: P \to \Lambda$ is a mapping called the labeling of the nodes of G.

$g: A \to Z$ where $Z=\{1,2,\ldots,d\}$ is a labeling of the edges of G.

M is a finite state automaton (Q,δ) where δ is a transition from $Q \times Z^t \times Q^t$ into Q in the deterministic case; $Z=\{1,2,\ldots,d\}$; $t \leq d$ is the degree of a node in G.

H is a mapping from P into Z^d. The image $H(n)=(i_1,i_2,\ldots,i_d) \in (Z \cup \{\#\})^d$ is called the <u>neighbor vector</u> of a node n. If n has degree $t \leq d$ then $i_{t+1} = i_{t+2} = \ldots = i_d = \#$.

Q is a finite set of states such that $\Lambda \subseteq Q$.



Informally there is an automaton M at each node n of G and the input state of the automaton at node n is $f(n) \in \Lambda$. At each time step, each M senses the states of its neighbors, reads its neighbor vector and changes its own state according to the transition function δ. The neighbor vector tells each node which neighbor of each of its neighbors it is.

A <u>cellular d-graph acceptor</u> is a cellular d-graph automaton $\bar{M}=(G,M,H)$ with a distinguished node D such that M is a finite state acceptor (Λ,Q,δ,F), where Λ (or Q_I) is the set of input states, (Q,δ) is a finite state automaton and $F \subseteq Q$ is a set of final states.

An <u>initial configuration</u> is a mapping $P \rightarrow \Lambda$. A <u>terminal configuration</u> is a mapping $c:P \rightarrow Q$ such that if n is the distinguished node D then $c(n) \in F$.

$\bar{M}=(G,M,H)$ accepts the d-graph $G=(P,A,f,g)$ if there is a finite sequence of configurations $c_0, c_1, \ldots c_m$ such that c_0 is an initial configuration, c_m is a terminal configuration and $c_i \vdash c_i+1$ for $0 \leq i < m$.

A <u>cellular d-graph automaton with augmented memory</u> is a graph G as above with each automaton M at each node of G having O(log N) memory, where N is the number of nodes in G. This amount of memory is sufficient to store numbers as large as N.

2. Labeling and counting

2.1 Numbering the nodes of the graph

Let G be a d-graph automaton with bounded degree d and let t be its minimum degree.

Lemma 2.1: The height of a breadth-first spanning tree T of G is bounded by $\log_{t-1}N$, where N is the number of nodes in G.

Proof: Such a tree is built by starting at the root node (D) and expanding all the nodes reachable by one edge from D. There are at least t such nodes. In parallel each of these nodes expands all its direct descendant nodes; ties are broken arbitrarily. At this and subsequent steps, each node has one ancestor and at least t-1 sons. The process terminates when it reaches leaf nodes, i.e., no more nodes can be expanded. Denote the number of expansion levels in G by h. Then

$$N=1+t+t(t-1) + t(t-1)^2+ \ldots + t(t-1)^{h-1} =$$

$$= 1 + \frac{(t-1)^h-1}{t-2} \cdot t \qquad \text{for } t\geq 3$$

Solving for h we get $h=\log_{t-1}[1+(N-1)(1-\frac{2}{t})]<\log_{t-1}N$

For t=d, $\log_{d-1}N$ is an approximate lowest value of h. In subsequent analysis we will assume t=d.

Consequence: It takes at most $2\log_{d-1}N$ steps for a signal starting at one node in G to reach any other node in G. This is a tight estimate if G≡T and one leaf node of T sends a signal to another leaf node on the opposite side of T.

Algorithm 2.1: Parallel numbering of the nodes in the graph.

(a) Construct a breadth-first spanning tree T of G.

(b) Store at each node the number of nodes in the subtree rooted at it. This procedure starts at the leaf nodes of T: each of them passes 1 to its ancestor. At the next time step each ancestor node accumulates the numbers received from all its sons, adds 1 (counts itself), stores the result in its internal memory and sends it up to its own ancestor. This process continues for $\log_{d-1} N$ time steps. By then D (the root of T) has gotten the total number of nodes in the graph.

(c) The labeling procedure is based on preorder tree traversal. Initially D is given the labels [1,N]. D labels itself with 1 and passes the labels [2,N] to its descendants $x_1,\ldots,x_d$ as follows: Suppose the numbers of nodes in the subtrees rooted at $x_1,\ldots,x_d$ are $n_1 \ldots n_d$. Then x_1 is given the labels $[2,n_1+1]$, x_2 is given the labels $[n_1+2,n_1+n_2+1]$,..., and x_d is given the labels $[N-n_d+1,N]$. Each x_i labels itself with the first label in its interval and divides the remaining interval of labels among its sons. This process is repeated recursively until it terminates at the leaf nodes of T.

Claim 2.1: Algorithm 2.1 labels the nodes of G in $O(\log_{d-1} N)$.

Proof: Constructing a breadth-first spanning tree takes $O(\log_{d-1} N)$ in parallel. Finding the number of sons at each node is done in parallel in $O(\log_{d-1} N)$ time as stated previously.

According to Lemma 2.1 the height (depth) of the tree T is $O(\log_{d-1} N)$, so that it takes a total of $O(\log_{d-1} N)$ to construct the labeling.

Both steps (b) and (c) in Algorithm 2.1 need O(log N) memory at each node.

2.2. Counting the edges and nodes in the graph

Algorithm 2.2: Edge counting

(a) Construct a breadth-first spanning tree T of G where each node marks its ancestor.

(b) The automaton at every node of T counts the number ℓ_n of edge emanating from n.

(c) Each leaf node of T passes its ℓ_n value up to its ancestor. Each such ancestor sums up all the values entering it and passes them up to its own ancestor. This process continues until D, the root of T, receives the sum of the ℓ_n values.

(d) D outputs half of this sum.

Claim 2.2: Algorithm 2.2 outputs the number of edges of G after at most $O(\log_{d-1}N)$ time.

Proof: Constructing a breadth-first spanning tree in parallel takes $O(\log_{d-1}N)$ time. As in Lemma 2.1 the depth of T is at most $\log_{d-1}N$. Therefore this is the time needed for the ℓ_n values to be summed at D. Algorithm 2.2 counts every edge of G twice: once for each of its endpoint nodes. Therefore D outputs half of the sum computed by it. Since there are at most $\frac{dN}{2}$ edges in a d-graph, each node needs no more than $O(\log_2 N)$ memory.

The procedure for labeling the nodes in G (Algorithm 2.1) also defines a labeling of the edges, since each edge is uniquely defined by the labels of its two endpoint nodes.

Algorithm 2.3: Node counting

(a) Construct a breadth-first spanning tree T of G.

(b) Each leaf node of T passes up the number 1 to its ancestor. Each ancestor addes 1 to the sum of the numbers it gets from all its sons and passes it up to its ancestor. This procedure continues in parallel for $\log_{d-1}N$ time steps until D receives the numbers from all its sons.

(c) D outputs the number of nodes in T (or G).

Claim 2.3: Algorithm 2.3 counts the nodes in G in $O(\log_{d-1}N)$ time.

Proof: Each node in T (or G) contributes 1 to the sum accumulated at D. According to Lemma 2.1, it takes $O(\log_{d-1}N)$ time for all the numbers to accumulate at D.

[1] also presented an algorithm for counting the number of nodes in G by passing up one bit for each node up in T and outputting the string of bits out of D. That scheme does not need augmented memory but avoids doing the summing in T. Counting the number of edges can be done in a similar way.

3. Cut nodes and bridges

3.1 Cut nodes

Let G be a d-graph labeled by Algorithm 2.1. Let T be the breadth-first spanning tree of G and D its root node.

Lemma 3.1.1: A leaf node of T cannot be a cut node of G.

Proof: Suppose X is a leaf node of T and a cut node of G. Then removing X would separate G into two connected components. D is in one of them (see Figure 1). Thus when we construct T starting at D, X must be expanded prior to any node in the component which does not include D. This contradicts the fact that X is a leaf node.

Consequence: There is no need to check leaf nodes for being cut nodes.

The cut node detection algorithm is based on the following steps:

Step (1): Label the nodes of T in a preorder traversal much like that in Algorithm 2.1 except that this time each node n is labeled with a pair of numbers $[\ell,m]$; ℓ is the lowest label given by Algorithm 2.1 to the subtree rooted at n and m is the highest label given to that subtree. Clearly the leaf nodes of T get labels of the form $[\ell,\ell]$.

Step (2): Upon being labeled each leaf node sends its own label to all its neighbors. Every ancestor of a leaf node accepts those labels coming from all its sons, calculates a new

pair $[\underline{\ell},\underline{m}]$ and sends it to all its neighbors at the next time step. In general every node in G is doing the same whenever it gets pairs of numbers coming from all or part of its neighbors except its father in T. The calculation of the pair $[\underline{\ell},\underline{m}]$ at each node is done as follows: Suppose node n in T is labeled $[\underline{\ell}_1,\underline{m}_1],[\underline{\ell}_2,\underline{m}_2],\ldots,[\underline{\ell}_{d-1},\underline{m}_{d-1}]$ from its d-1 neighbors (not including its father in T). Then n computes $\underline{\ell}=\min\{\ell,\ell_1,\ell_2,\ldots,\ell_{d-1}\}$ and $\underline{m}=\max\{m,m_1,m_2,\ldots,m_{d-1}\}$ and sends the pair $[\underline{\ell},\underline{m}]$ to all its neighbors. Upon acceptance of the pairs $[\underline{\ell},\underline{m}]$ every node (except the leaf nodes of T) compares the pairs received from each of its sons in T to its own stored label $[\ell,m]$ and also to the label of that son in T, by sensing that label. This comparison, described in Lemma 3.1.2, gives each node an indication of whether it is a cut node based on the current information at the node. The above process of sending pairs (the signaling process) continues for $\log_{d-1}N$ time since this is the maximum time needed for the root node D of T to be influenced by the signals coming from the leaves of T. By then each node is capable of deciding whether it is a cut node by consecutive examination of the labels entering it from its sons in T. Leaf nodes of T do not make the comparisons of the following Lemma 3.1.2 since they are not candidates for being cut nodes by Lemma 3.1.1. However, they do participate in the signaling process.

Lemma 3.1.2: A node n in a d-graph G is a cut node if and only if one of the following conditions holds:

1) No label received by n during step (2) from any of its direct descendants in T is outside the range defined by the label $[\ell,m]$ of n. In this case removal of n will separate the subtree rooted at n from the rest of G.

2) The labels received from at least one of n's direct descendants in T, during step (2), are within the range of the label $[\ell,m]$ of that descendant. In this case if n is removed, the subtree rooted at that descendant will be disconnected from the rest of G.

Both conditions 1) and 2) can be satisfied at a node n. In that case G is divided into more than two connected components upon removal of n.

Proof: (1) Let $[\ell,m]$ denote the label of the node n.

if: During the signaling process if n does not get from its sons any pair of labels which is outside the range $[\ell,m]$, then n is not connected, via its descendants in T, to any node of G outside its subtree and thus is a cut node.

only if: If node n' having subtree S1 (Figure 2) is not a cut node, i.e., is connected in G to node n* outside the subtree S of n, then the signaling process would cause node n* to send its $[\underline{\ell}(n^*),\underline{m}(n^*)]$ pair to node n'. This pair is outside the range of labels for S1, due to the preorder labeling of step (1). Therefore one of these pair elements will become

$\underline{\ell}(n')$ or $\underline{m}(n')$ of the pair to be conveyed to n by the signaling process. Thus n will receive a label outside the range of its own label $[\ell,m]$ contradicting condition (1) of the lemma.

The condition of comparing only pairs received from direct descendants is necessary since if n also compares pairs coming from directly connected non-sons it might conclude incorrectly that it is not a cut node. Note that nodes connected directly to n which are not its direct sons in T will always send it pairs whose ranges do not overlap $[\ell,m]$.

(2) if: Suppose the pair $[\underline{\ell},\underline{m}]$ received by n from S1 (Figure 2) in G is within the range $[\ell(n),m(n)]$. This means that S1 is not connected to the other nodes of G except via n because of the preorder labeling of T. Thus n is a cut node with respect to S1 in G.

only if: If n is not a cut node with respect to S1 (Figure 2), then S1 is connected to S2 or S3 or the rest of G and thus during the signaling process will convey to n a pair $[\underline{\ell},\underline{m}]$ which is outside the range $[\ell,m]$ stored at n' in violation of condition (2).

Algorithm 3.1.1: Finding cutnodes of G

(a) Construct a breadth-first spanning tree T of G.

(b) Use step (1) to label all the nodes of G.

(c) Apply steps (2) during which each node of G decides whether it is a cut node according to case (1) or case (2) of Lemma 3.1.2.

Claim 3.1.1: Algorithm 3.1.1 finds all the cut nodes in G in time $O(\log_{d-1} N)$ and with O(log N) storage per node.

Proof: Time: Steps (a) and (b) take $O(\log_{d-1} N)$ time. Step (c) needs $\log_{d-1} N$ time for the propagation of signals. Constant time is spent for comparisons at the nodes.

Memory: The amount needed at each node for storing the labels $[\ell, m]$ and $[\underline{\ell}, \underline{m}]$ is 4log N.

3.2 Bridges

Let G be a d-graph labeled by Algorithm 2.1 and T be its breadth-first spanning tree.

Lemma 3.2.1: Every bridge of G is in T.

Proof: Let x be a bridge of G and S the set of edges of T. if $x \notin S$ then removing x from G will not disconnect it and thus x is not a bridge of G.

The bridge detection algorithm is based on the following steps:

Step 1: The same as step (1) of Section 3.1.

Step 2: The signaling process here is much the same as that in Section 3.1 except that the comparison procedure is done bewteen all the pairs entering a node n and the label $[\ell,m]$ of that node. No direct comparison is done with the labels of n's sons. The signaling process of sending $[\underline{\ell},\underline{m}]$ out of each node is the same as in Section 3.1.

Lemma 3.2.2: A bridge x connecting a node n to its direct ancestor in T is a bridge of G if and only if the following condition holds: No label received by n through any of its edges, except x, is outside its own stored label $[\ell,m]$.

Proof: Similar to case (1) of Lemma 3.1.2. Clearly if labels are allowed to enter n via x, then n will always receive labels outside its own label range and will conclude that x is not a bridge even if x is the only edge connecting n and its subtree to the rest of G.

Algorithm 3.2.2: Finding bridges of G

(a) Construct a breadth-first spanning tree T of G.

(b) Use step (1) to label the nodes of T.

(c) Apply steps (2) during which each node n of T decides, according to Lemma 3.2.2, whether the edge connecting it to its ancestor in T is a bridge.

Claim 3.2.2: Algorithm 3.2.2 finds all the bridges in G in $O(\log_{d-1} N)$ time and O(log N) memory per node.

Proof: Time: Steps (a) and (b) take $O(\log_{d-1} N)$ time each. Step (c) needs $\log_{d-1} N$ time for the propagation of labels. The computation at the nodes takes constant time.

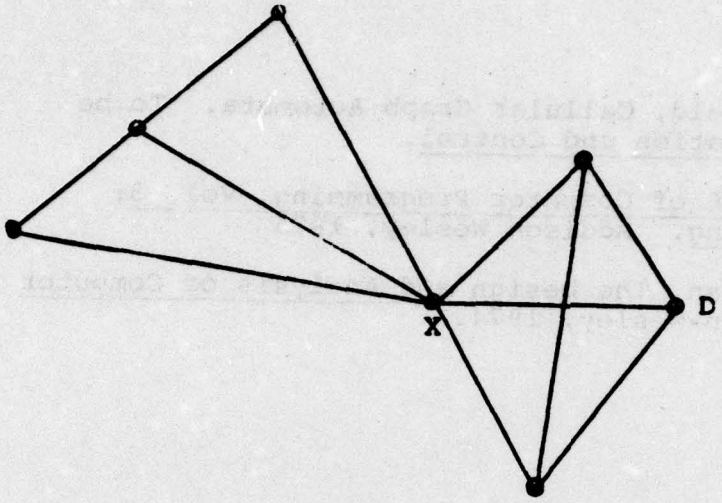
Memory: Each node needs 4log N to store its own label and $[\underline{\ell},\underline{m}]$.

References

[1] A. Wu and A. Rosenfeld, Cellular Graph Automata. To be published in Information and Control.

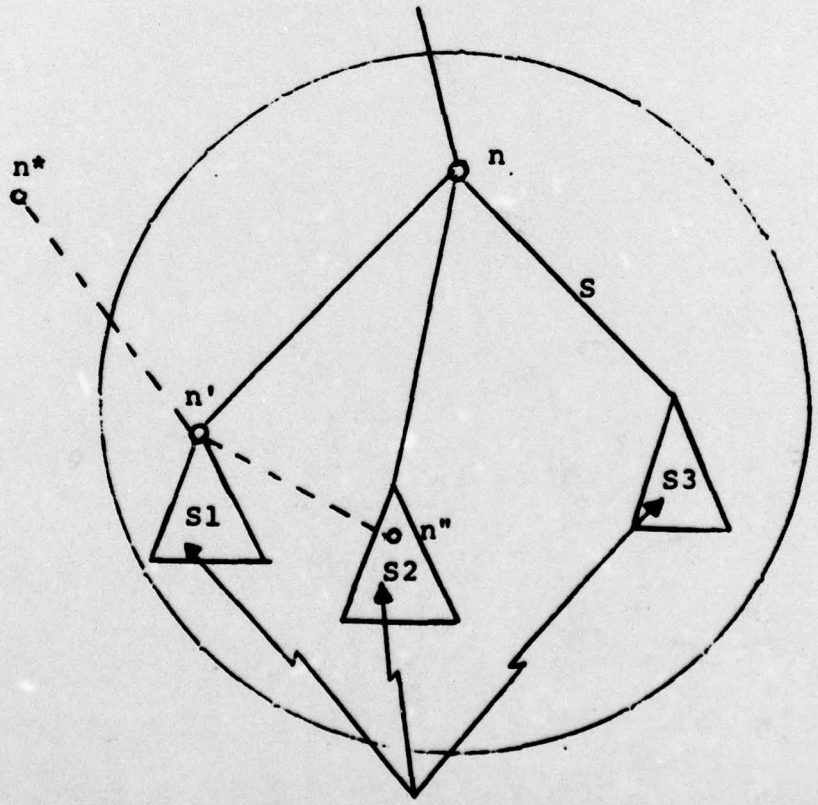
[2] D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison Wesley, 1975

[3] Aho, Hopcroft, Ullman, The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.



G

Figure 1



subtrees of n in T

Figure 2

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR-79-1164	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): CELLULAR D-GRAPH AUTOMATA WITH AUGMENTED MEMORY,	5. TYPE OF REPORT & PERIOD COVERED: Interim
6. PERFORMING ORG. REPORT NUMBER: TR-771	
7. AUTHOR(s): Tsvi Dubitzki, Angela Wu	8. CONTRACT OR GRANT NUMBER(s): AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Center, University of Maryland, College Park, MD 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304 A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB, Wash., DC 20332	12. REPORT DATE: Jun 79
	13. NUMBER OF PAGES: 16
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Cellular automata
Graphs

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper deals with cellular d-graph automata in which each node has internal memory proportional to $\log_{d-1} N$ where N is the number of nodes in the underlying graph G. It is shown how such an automaton can assign unique labels to the nodes of G in $O(\log_{d-1} N)$ time. Such an automaton can also count the number of nodes and edges in G in → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 403 028

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract continued.

O($\log_{d-1} N$) time. Algorithms for identifying all the cut nodes and all bridges in G, each in O($\log_{d-1} N$) time, are also presented.

sub d-1 of