PATH SEARCHING IN SWITCHING NETWORKS USING CELLULAR ALGORITHM

By

L. T. KÓCZY, J. LANGER and T. LEGENDI

Department of Telecommunication, Technical University, Budapest.

Received February 9, 1981

Presented by Prof. Dr. S. CSIBI

1. Introduction

One of the most time consuming jobs in stored program control of telephone exchanges is path searching, at least when the structure of the switching network is complicated enough (as it is in practically all larger exchanges). Switching fields are always constructed so, that for two marginal points given in the graph of the network, numerous correct possibilities exist for determining a path connecting them. When the exchange is in function, however, not all theoretical possibilities come into consideration: no path which contains any edge in the graph, already used in another path (belonging to an existing speach connection) is allowed. It is even possible that in a given state there is no path allowed at all, a fact resulting in a virtual busy state of the called subscriber. Path searching must be carried out in a very short time, as after dialling the caller person gets impatient if the ringing (or busy) tone has not arrived in about a second. It is also possible, that virtually at the same time hundreds or thousands of subscribers initiate a call, so at least hundreds of paths must be determined in a second — beside a lot of other real time functions of the control processor.

For fulfilling the above requirements, it was necessary in all existing larger exchanges to form the structure of the switching network in such special way that quick and effective algorithms could be found for path searching in them.

This method, however, has also disadvantageous properties: the network cannot be designed directly on the basis of traffic and other technical conditions: it must fulfill simultaneously partly contradictory requirements, resulting in much additional costs. Another problematic situation is when the structure of network cannot be changed: It is e.g. given by the necessary combination of different exchanges. In all cases where the complicatedness of path searching causes a time consumption higher than allowed by the real time conditions, the opportunistic solution must be accepted, where searching is stopped when a certain time limit has been reached. Then, although it is possible that there exists a convenient path, a busy tone must be given out.

Motivated by the above considerations, we started to look for an algorithm general and quick enough to be used in practice in switching networks of arbitrary structure.

2. The problem of path searching

Let us consider a simple 4 stage switching network (its graph see in Fig. 1). Path searching has been already carried out from the caller switching matrix $A1$ and the called matrix $A4$ until stage D, also the free DD links have been detected. In the figure only the free paths running uninterruptedly from $A1$ to

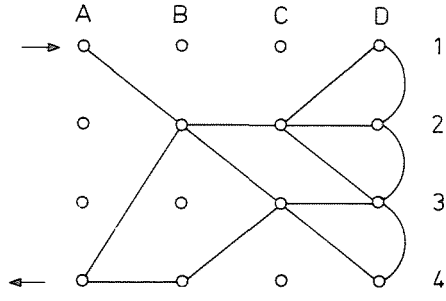

Fig. 1

$A4$ are marked. Now we use e.g. the minimum index principle when choosing a single free path:

$$A1-B2-C3-D1-D2$$

from the caller. Now to the called matrix only one possibility remains:

$$D2-C2-B2-A4\,.$$

By this we have chosen link $B2-C2$ twice, but this fact has not been yet detected by the processor. In order to eliminate double selection, the algorithm must check all pairs of links between the same stages. After the detection of the forbidden selection a new $A-D$ path must be chosen:

$$A1-B2-C2-D2-D1\,.$$

The returning $D-A$ path will be now

$$D1-C2-B2-A4\,.$$

Here the same type of failure occurs. The next version is:

$$A1-B2-C2-D2-D3-C2-B2-A4\,.$$

Failure. In our very simple example only the forth version will be sufficient:

$$A1-B2-C2-D2-D3-C3-B2-A4\,.$$

As a matter of course, by using another principle in choosing the paths — e.g. the maximum index principle or random choice — much earlier a "good" path would have been found in this case. It is clear, however, that repeated checking of paths and possible rejections consume a considerable amount of time.

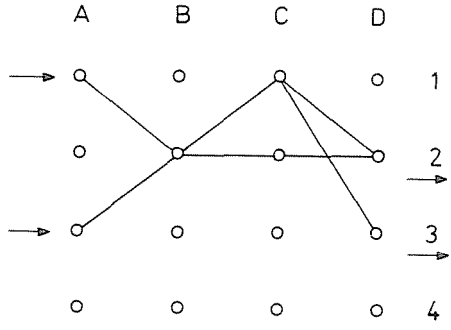

Fig. 2

Let us consider now a second example — for another type of path searching problems: Two demands of connection have been raised and searching must be carried out practically at the same time. See e.g. the "one direction" network of Fig. 2 ($A1$ to $D2$ and $A3$ to $D3$). After the examination of link conditions we choose for the first connection

$$A1-B2-C1-D2\,.$$

So in the situation of Fig. 2 the second demand must be rejected. In the case, however, when both demands are considered simultaneously and instead of the above one, the following is chosen:

$$A1-B2-C2-D2\,,$$

there remains a free path for the second connection, too:

$$A3-B2-C1-D3\,.$$

It is again very difficult to check all alternative possibilities for one or several already existing connections, in order to gain a re-arrangement strategy — carrying out these examinations one after another.

In both the last examples the main problem is common: how to find at least two disjoint paths in the same network, at the same time. The problem in a folded network (like in Fig. 1) is also very similar, as the two main parts of a path — namely, the foregoing and the returning parts — can be considered as two different foregoing paths in a simple network — only they have to reach a pair of matrices in the last stage, which is connected by a free link. In [1] we have given a transformation rule for mapping the above problems always to a simple foregoing path searching job — although in a more complicated network. In the next section we are giving a brief summary of this transformation and then a general path searching algorithm which can always be used for solution of the transformed problem. Then, in Section 4, we show the way of realizing it by cellular automata.

3. The path searching algorithm

A graph will be named a switching network graph (SNG) if there is a subset of its vertices (start vertices) so, that a) there is no path of length 1 between any pair of them, b) there exists another subset of vertices (termination vertices) so, that there is at least one continuous path between an arbitrary pair of vertices taken from these two subsets. (It is not necessary, that these two subsets be disjoint or different at all.)

A graph is named a simple SNG if the start and termination subsets are disjoint.

In an SNG we name a set of paths a switching path bundle if all paths start and terminate in the start and termination subsets and all paths are disjoint pair by pair.

A simple switching path (SSP) is a SP in a simple SNG if the number of paths in it is 1.

If given an SNG G, there exists an invertible mapping M, so that $H = M(G)$ is a simple SNG and for all SP-s P in G, if the number of paths in P is not greater than a given n, $R = M(P)$ is SSP in H. The above statement is very general, so it cannot be used practically in this form. However, for the two main problems mentioned in Section 2, rather simple implementations can be given. In order to present them, some further definitions must be introduced:

If the vertices of a simple SNG can be partitioned into $n > 2$ subsets:

$$S_1, \ldots, S_n (S_i \cap S_j = \emptyset \qquad \text{if} \qquad i \neq j$$

and $\bigcup_{i=1}^{n} S_i = V(G)$, the set of vertices in the graph), so that there is no edge between any pair of vertices taken from the same S_i, the graph is named a staged one direction simple SNG (DG). (S_1 is the start, S_n the termination subset.)

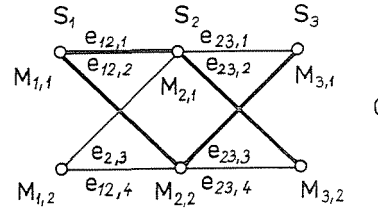

Fig. 3

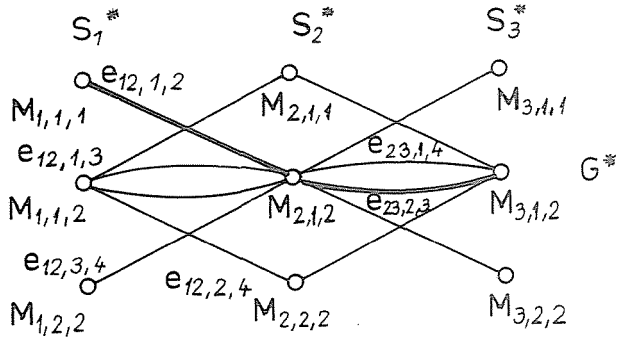

Fig. 4

If given a DG G, there exists the mapping M, according to the statement, so that $H = M(G)$ is also DG, the number of stages n in G and H is equal, and if $m_{i,j}$ is the number of edges between stages S_i and S_j in G, in H the same numbers are

$$m'_{i,j} = \binom{m_{i,j}}{k},$$

where k is the maximal number of paths in the SP-s in consideration in G. For a proof and construction rule of H see [1]. As illustration to the above, in Fig. 3 we give a simple SNG, where $n=3$, $S_i=2$ for $i=1, 2, 3$ and $k=2$. The corresponding $H=M(G)$ is to be seen in Fig. 4.

A similar statement can be given for the folded network problem. First we have a definition. If an SNG can be partitioned in $S_1, \ldots, S_{n-1}$ forming a DG and into

$$S_n(S_n \cap S_i = \emptyset \qquad \text{if} \qquad i = 1, \ldots, n-1$$

and $\bigcup_{i=1}^{n} S_i = V(G)$), so that S_n contains only vertices connected at least with one vertex in S_n (including the possibility of being connected with itself) and one vertex not in S_n; G is named a staged folded SNG (FG).

If given an FG G, there exists the mapping M according to the first statement, so that $H = M(G)$ is DG with n stages and

$$m'_{i,j} = \binom{m_{i,j}}{2k},$$

(notations as before).

Proof and construction rule can be found in [1]. Again, we give an example by the simple Figs 5 and 6 (original and transformed network,

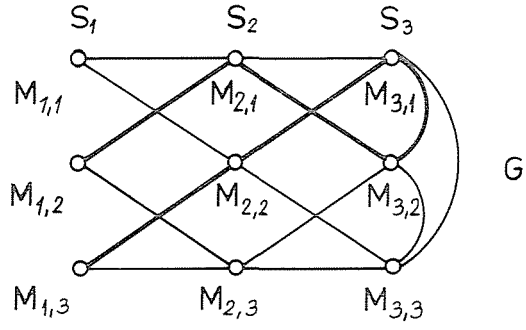

Fig. 5

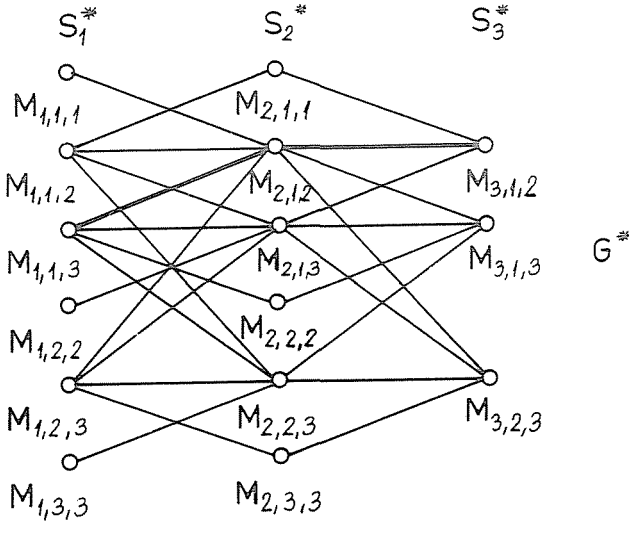

Fig. 6

respectively). By this method, all practically important problems were reduced to the searching of an SSP in a DG, although the number of links between two stages would be multiplied by at least

$$\binom{m}{2}/m = (m-1)/2$$

(m is the number of links between two stages in the original network). Similarly, the number of vertices increases too.

We have eliminated complicated path bundles, now we concentrate on the irregularity of networks, but consider only SSPs in DGs. The statements before allow this restriction.

What are the main features of an irregular network?

1. The dimensions of the switching matrices (SM) are different even in the same stage.

2. Not all cross points exist really (it can be even time variant because of failures).

3. Some links connect not adjoining stages. (We shall name them passing links.)

In our algorithm we want to model switching matrices by binary incidence matrices. Different SMs belonging to the same stage can be merged into one virtual stage SM (having no switch at certain cross points of in- and outgoing links). If there is no switch at a certain crosspoint (e.g. as the in- and the outgoing links belonging to this point do not belong in reality to the same SM, or because of point 2), the value in the incidence matrix must be fixed as "0". Else, the value is "1". Passing links are considered as two (or more) links connected by 1×1 SMs (according to each stage, where they are passing by). By these rules points 1—3 are eliminated as problems when each stage is modelled by a single binary matrix. (See e.g. Fig. 7.)

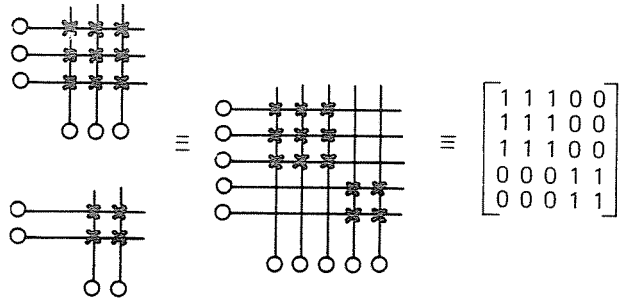

Fig. 7

Additionally, the permutation structure of the links between adjoining stages can be characterized by ordinary permutation matrices. One stage matrix (as above) and one permutation matrix (of the links) together form a

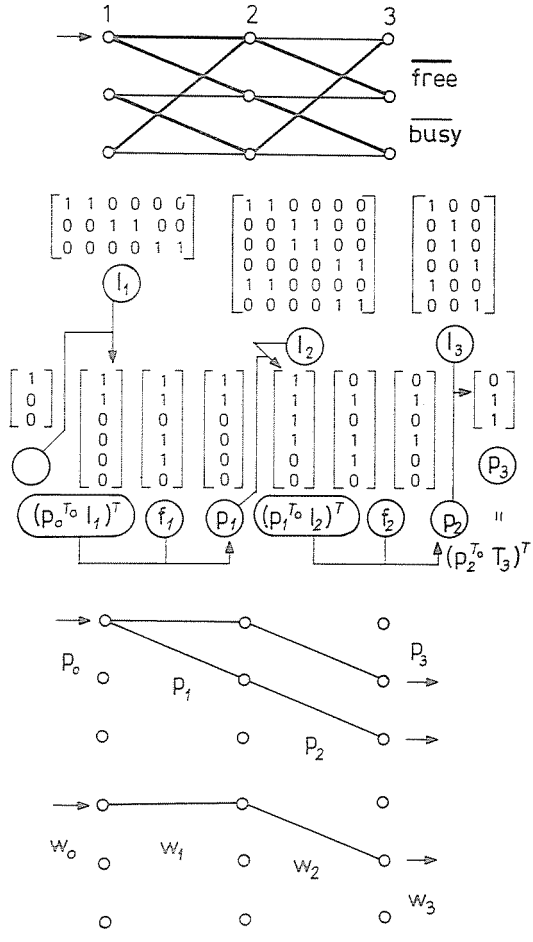

Fig. 8

structure characteristic matrix, which is virtually stable. An arbitrary DG can be described by n matrices: $I_1, \ldots, I_n$. Now, the states of links at a given time can be described by a set of vectors: $f_1, \ldots, f_{n-1}$, which contain "1"-s at the positions where the corresponding link is free and "0"-s where it is busy. The path searching problem can be defined by f_0 and f_n indicators, which mark the allowed start and termination points, respectively by "1"-s, in the case of a DG obtained by the transformation of an FG, however, f_n is the state vector of the returning links (in S_n) in the original FG.

Given the above data, the path searching algorithm is as follows:

$$p_0 = f_0$$
$$p_i = (p_{i-1}^T \circ I_i)^T \wedge f_i \qquad \text{for} \quad i = 1, \ldots, n$$
$$w_n = p_n$$
$$w_i = (I_{i+1} \circ w_{i+1}) \wedge p_i \qquad \text{for} \quad i = n-1, \ldots, 0$$

(o stands for logical matrix multiplication.)

Here, w_i $(i = 0, \ldots, n)$ presents the subgraph containing all free paths from the start vertices to the termination vertices indicated by f_0 and f_n, respectively. Illustration of the algorithm is shown on Fig. 8. For further details of the matrix algorithm see [2] and [1].

By the sketched method we have an algorithm for finding parallelly and "good" paths in a simple SNG. The method operates only with simple logical operations, but even so, because of the high number of steps, time consumption can be enormous. When the original problem is complicated enough, and a previous application of the graph transformation is necessary, the dimensions of the incidence matrices grow considerably. This is a serious difficulty with the practical application. So, in the next section we present a method by which time consumption can be reduced to practically acceptable extent.

4. The algorithm in cellular fields

In the next part we are sketching the realization of the above matrix algorithm by cellular automata. We shall use homogeneous cellular fields built up of 5 neighbour 8 state automata first suggested by Codd [3].

As a first example, it is intended to demonstrate, how easily the composition of binary matrices can be solved by such cellular fields. The composition (o) is defined by:

$$[c_{ik}] = [a_{ij}] \circ [b_{jk}] \qquad \begin{matrix} i = 1, \ldots, I \\ j = 1, \ldots, J \end{matrix}$$
$$c_{ik} = \bigvee_j (a_{ij} \wedge b_{jk}) \qquad k = 1, \ldots, K$$

We use a cellular field of dimension $I \times K$. (Plus the marginal cells.) The state of the cell with coordinates x, y is

$$Q_{x,y} = (q_{x,y}, r_{x,y}, s_{x,y}) .$$

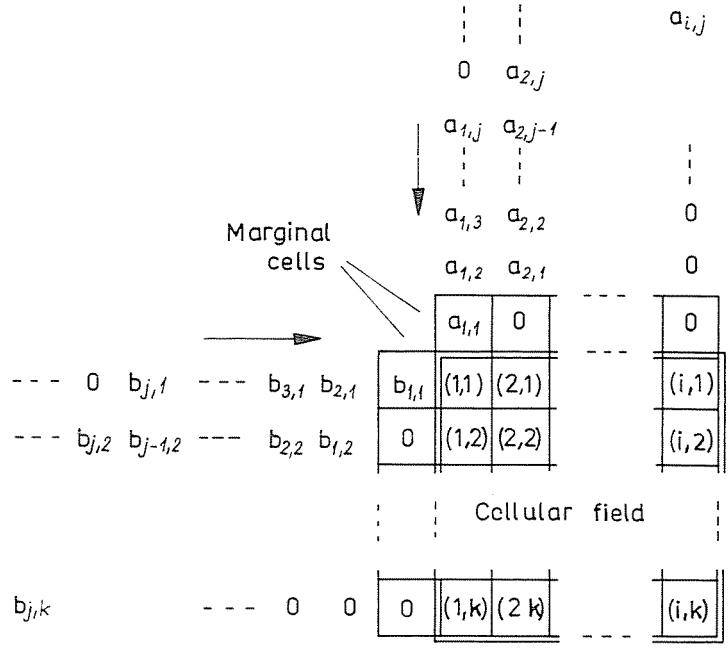

Fig. 9

Then the algorithm is defined as follows:

$$Q_{x,y}(T=0)=(0,0,0) \qquad x=1,\ldots,I;\ y=1,\ldots,J$$

$$Q_{x,y}(T\neq 0)=(q_{x-1,y}(T), r_{x,y-1}(T), s_{x,y}(T)\vee(q_{x,y}(T)\wedge r_{x,y}(T)))$$
$$x=1,\ldots,I;\ y=1,\ldots,J$$

For $Q_{0,y}(T)$ and $Q_{x,0}(T)$ see Fig. 9. After the last step c_{ik} is to be found in the cellular field.

Above process was only an illustration, as the path searching algorithm itself needs no repeated composition of real matrices (only if failures occur in the switching matrices). Needed are the following four operations:

A) Composition of a vector and a matrix
B) Conjunction of two vectors
C) Transformation of vectors into unit vectors (only for path selection)
D) Transponation of vectors or matrices

Operation A) is a special case of the matrix composition, it can, however, be realized easier by a somewhat modified algorithm and in a cellular field $I\times K$. The algorithm is as follows (the vector $[a_k]$, the matrix $[b_{jk}]$):

$$Q_{1,y}(T=0)=(0,0,0)$$

$$Q_{1,y}(T\neq 0)=(q_{1,y}(T)\vee(q_{0,y}(T)\wedge q_{2,y}(T)\,(0,0,0) \qquad y=1,\ldots,K$$

For $Q_{0,y}(T)$ and $Q_{2,y}(T)$ see Fig. 10. For completion of the operation, K number of steps are needed.

Operation B) is very simple, let us take e.g. two vectors $[a_k]$ and $[b_k]$, then

$$Q_{1,y}(T=0)=(a_y,0,0)$$

$$Q_{1,y}(T=1)=(q_{1,y}(T=0)\wedge q_{0,y}(T=0)) \qquad y=1,\ldots,K$$

if $\quad Q_{0,y}(T=0)=(b_y,0,0)$.

It is completed in 1 step.

Operation C) cannot be solved economically even by cellular automata, it needs K number of steps for $[a_k]$.

Finally, by convenient data transfer, D) can be eliminated totally as an operation to be counted with.

Let us summarize now, how to construct the full algorithm:

Given a stage in the switching network, described by an $n\times m$ matrix (I_i) and a state vector $(f_i)\cdot p_{i-1}$ has been already computed. Then, in order to obtain p_i, we have to perform the following operations:

$$A(p_{i-1},I_i): \qquad m \text{ steps}$$

$$D(p_{i-1}^T\circ I_i): \qquad 0 \text{ steps}$$

$$B(p_{i-1}^T\circ I_i,p_i): \qquad 1 \text{ step}$$

Needed are $m+1$ steps together. For preparing p_i for storing one additional step is necessary, so the total number of steps is $m+2$. Similarly, if given I_i, p_{i-1}

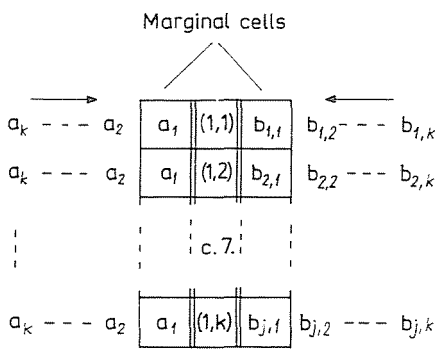

Fig. 10

and w_i, for obtaining and storing w_{i-1} $n+2$ steps are needed. If we have a switching network with stages s (dimensions of stage i are $n_{i-1} \times n_i$) the total number of necessary steps is:

$$L_p = \sum_{i=1}^{s} (n_{i-1}+2) + \sum_{i=1}^{s} (n_i+2) = 4s + n_0 + n_s + 2 \sum_{i=1}^{s-1} n_i .$$

In the simple case, when $n_i = n$ for all i,

$$L_p^{hom} = 4s + 2sn \approx 2sn .$$

Let us examine now, how many steps are needed if a serial processor is used:

For operation A), needed are $2nm - n$ steps, for operation B), m steps, for storing p_i also m steps. So the total amount of necessary steps with a serial processor is:

$$L_s = 2 \sum_{i=1}^{s} n_{i-1}(n_i + 1) - \sum_{i=1}^{s} n_i + 2 \sum_{i=1}^{s} (n_{i-1}+1)n_i - \sum_{i=1}^{s} n_{i-1} =$$

$$= 4 \sum_{i=1}^{s} n_{i-1} n_i + 2 \sum_{i=1}^{s-1} n_i + n_0 + n_s .$$

In the special case:

$$L_s^{hom} = 4sn^2 + 2sn .$$

The time consumption rate between serial and parallel (cellular) processing will be e.g. in the special case discussed above:

$$R^{hom} = \frac{L_s^{hom}}{L_p^{hom}} = \frac{4sn^2 + 2sn}{2sn + 4s} \approx 2n .$$

As in practical cases n's value is between 100 and 100,000 (for the transformed graphs), by using cellular algorithm an enormous saving of time can be reached.

We did not take into consideration operation C) when counting the number of steps. It must be done in the second run, when w_i-s are computed, so to L_p $\sum_{i=0}^{s} n_i$ must be added. The same steps are needed in the serial case, so the real value of R will be lower, e.g.:

$$R^{hom}, \mathrm{SP} = \frac{4sn^2 + 3sn}{3sn + 4s} \approx 4/3n\,.$$

It is to be noted, that operation C) is needed only in the case, when at least one free path exists. In general, R^{hom}-s practical value is somewhere between n and $2/3n$. (It depends on traffic conditions.)

5. Final remarks

Some illustrative simulation examinations of the algorithm [4] have been carried out using a special simulation language developed in our country [5], on a computer CDC 3300.

We intend to go on with further, practical examinations, where traditional serial control processor(s) will be connected with cellular processor(s) used for path searching and maybe some other function; so it is tried to gain a quick and generally applicable control system for switching exchanges.

Summary

After a survey of the important statements in the paper "A mathematical model of path searching in general type switching networks" (appeared formerly in the journal) this paper treats with possible implementation for cellular automata of the algorithm introduced there. The cellular field used here consists of 5 neighbour 8 state cells.

The final statement of the paper concerns the comparison of running times required by a traditional serial processor and by the cellular field, respectively: by parallel processing this running time can be reduced to its fracture.

References

1. KÓCZY, L. T.: A mathematical model of path searching in general type switching networks, (to appear in Periodica Polytechnica).
2. KÓCZY, L. T.: Ein Wegesuchalgorithmus für Durchschaltenetzwerke. 23. Internationales Wissenschaftliches Kolloquium der TH Ilmenau, GDR, Vortragsreihe Nachrichtentechnik, pp. 133—136, (1978)
3. CODD, E. F.: Cellular automata, Academic Press, New York—London 1968.
4. LANGER, J.: Examination of the application of cellular processors in telephone engineering, MSEE work (consulted by L. T. Kóczy and T. Legendi), Dept. of Telecommunication, TU of Budapest, 1980.
5. MOLNÁR, G., SZÉKELY, L., LEGENDI, T.: CELLAS 1 0 user's manual, Szeged, 1979 and L. SZÉKELY: FA-1 function defining subsystem user's description, Szeged, 1979.
(Items [4] and [5] in Hungarian language.)

dr. László T. KÓCZY
József LANGER
dr. Tamás LEGENDI

H-1521 Budapest
BHG Telecommunication Works
Hungarian Academy of Sciences, Research Group on Automata